TREBALL FINAL DE GRAU

TÍTOL: DESENVOLUPAMENT D'UNA XARXA SOCIAL PER STARTUPS

AUTOR: TER CABALLERO, JOAN MARC

DATA DE PRESENTACIÓ: JULIOL, 2019

COGNOM: TER CABALLERO **NOM: JOAN MARC**

TITULACIÓ: GRAU EN ENGINYERIA INFORMÀTICA

PLA: 2010

DIRECTOR: NEUS CATALÀ ROIG

DEPARTAMENT: CIÈNCIES DE LA COMPUTACIÓ

QUALIFICACIÓ DEL TFG

TRIBUNAL

PRESIDENT	SECRETARI	VOCAL
MERENCIANO SALADRIGUES, JOSEP MARIA	MATA MIQUEL, CRISTIAN	GARCÍA ALMIÑANA, JORDI

DATA DE LECTURA: 8 DE JULIOL DE 2019

Aquest Projecte té en compte aspectes mediambientals: □ Sí ■ No

RESUM

Quan pensem en el concepte *Startup* el primer que ens ve al cap és un conjunt de persones que acaben de començar a desenvolupar una idea que, amb una mica de sort i un bon plantejament de com enfocar la situació, arribarà possiblement a tenir èxit i naixerà una empresa dins d'aquest sector que any rere any no atura el seu creixement. Per tal d'arribar a aquest punt, és necessari fer les coses d'una manera molt organitzada i tenir l'equip el més comunicat i compacte possible. D'aquesta manera, dissenyar una aplicació per aquestes futures empreses és l'objectiu principal d'aquest projecte.

Avui en dia hi ha moltes alternatives a l'hora d'escollir un software de comunicació, missatgeria, organització de tasques o esdeveniments. Per això, un dels objectius és poder agrupar totes aquestes necessitats que es busquen en altres aplicacions dins d'una única plataforma a través de la qual l'empresa pugui interactuar, treballar i organitzar la feina de la manera més eficient possible. Una aplicació capaç de poder gestionar converses, perfils d'usuari, publicacions dels usuaris, entre d'altres, per així no haver de centrar molts dels recursos en escollir les millors eines per gestionar el projecte i l'equip i dedicar-los a altres feines més importants que requereixen més atenció a l'inici d'una empresa.

Així doncs, a través d'un base de dades no relacional i una *API*, he desenvolupat aquesta aplicació web per assolir els objectius principals comentats. S'ha dissenyat l'arquitectura de tal manera que pugui ser fàcilment escalable i així no tancar-nos a futures ampliacions o millores que s'hi puguin crear. De la mateixa manera, a l'haver de crear una aplicació d'aquestes magnituds és molt important tenir molts clars els objectius, els requeriments i l'abast. La metodologia àgil aplicada en aquest desenvolupament ha estat un punt clau que m'ha permès quadrar els temps i les tasques perfectament, donant-me un marge de rectificació ampli i una estimació del progrés que ha facilitat molt tot el camí. Utilitzant SCRUM, la divisió dels *Sprints* ha estat de dues setmanes, fent que els blocs de feina es desenvolupessin ràpidament i així els he pogut testejar entre ells a cada final d'*Sprint*.

El resultat ha estat totalment satisfactori, l'aplicació ha complert tots els requeriments del client (fins i tot els que eren completament opcionals), oferint una primera versió del producte molt completa i amb les portes obertes a les futures millores que s'hi poden dissenyar.

Paraules clau (màxim 10):

Startup	Aplicació	Organització	Escalabilitat	Dades
Futur	Tecnologia	Equip	SCRUM	JavaScript

ABSTRACT

The Startup concept has been steadily gaining importance over this last decade and this caught me directly at the age where I started my first job. I have been working on Startups since the beginning, just after finishing my studies, and luckily (or maybe not) I have been able to experience all its positive and negative aspects on a personal level. That is why I thought that this could be a very good opportunity to invest in a tool that could help this future companies on its early stage to focus all the efforts on the really important things instead of looking for the right software tools to organize the team in order to grow faster.

Nowadays there are lots of alternatives when it comes to finding the right application that solves a particular need, as Gmail does for mailing purposes or Slack for communication among the team. All these tools are so popular and well-known that it is obvious that these would be the first ones that a new-born company would look at. But that is when this project comes in. The main reason behind this project is to build a web tool that helps and fulfills all those needs in just one application able to group all the daily tasks that each person in the company has to deal with every day. From messaging to creating events, personal posts and user profiles, it is meant to be designed in a way that everyone feels comfortable, spending less time on doing all the required things and therefore be more productive, as it is all in the same application.

The main requirements from the client were simple: people should be able to message among them, publish unlimited posts that could help others to learn new things, create events to strengthen the team and have a public profile for each member so that everyone could refer to. I have built a whole infrastructure, from a MongoDB database to a RESTFul API, ready to serve all the data to the brand-new web application called SmartNet. It has been designed in a way that it can be handled by any client to retrieve any data and build a new application on top of that, so it can scale and succeed in any new project that this tool can be the roots of.

The final result has been a staggering success. The application has fulfilled all the requirements from the client (even those that were completely optional), offering a first version of a very complete product and also the possibility to improve it by giving a complete customizable API.

Keywords (10 maximum):

Startup	Application	Organization	Scalability	Data
Future	Technology	Team	SCRUM	JavaScript

Gràcies al Carles Escrig i Royo, un gran amic de Castelló que em va animar a realitzar aquest projecte i al qual dec gran part del coneixement que tinc avui dia.

Índex

1. Introducció

1.1. Context

És curiós com avui en dia ha canviat tot el tema de les empreses i el sector tecnològic, sobretot pel que fa a la idea que tenia jo abans d'acabar la carrera i pensant en què m'esperaria quan entrés al món laboral. Pensant-ho en fred, mai hauria imaginat que a hores d'ara només hauria treballat a *Startups*, res de consultores o empreses grans, amb un nombre elevat de treballadors i unes oficines que no te les acabaries mai... O com a mínim això era el que imaginava fins que vaig sortir de la universitat.

Fa uns 3-4 anys que treballo al sector tecnològic, en concret a l'àmbit de desenvolupador web. He estat a dues empreses diferents, les dues *Startups*, una de Vilanova i la Geltrú i l'actual a Barcelona. Durant aquests anys ha canviat molt el pensament que jo tenia respecte a aquest tipus d'empreses, tant del que pensava respecte al tema de l'ambient dins d'un equip petit com del producte que s'acostuma a desenvolupar en aquests llocs, normalment un producte "innovador" i que encara no està llest o consolidat dins del mercat. Totalment al contrari.

Al mateix temps que m'anava fent la idea d'on era i com em replantejaria el futur després d'haver estat formant part d'un equip d'aquestes característiques, també vaig anar veient i coincidint amb forces coses que les dues empreses tenien en comú: falta consolidació en alguns aspectes dels projectes, ja sigui a nivell d'estructuració de l'equip com a nivell de l'organització de les tasques. Cal dir que tampoc no he vist altra cosa que no sigui un equip d'una empresa *Startup* i molt probablement això pugui passar a qualsevol altra empresa, però sí que és cert que té força sentit que s'apreciï tan fàcilment en equips petits que desenvolupen un projecte que és totalment nou i que encara no s'ha creat ni una primera versió. El mercat no coneix aquest producte, o com a mínim l'alternativa que s'està desenvolupant, cosa que fa que el procés hagi de ser relativament ràpid per ensenyar-ho al món i a vegades coses tant obvies com planificar un camí de tasques amb calma o saber escollir bé les eines pel desenvolupament o comunicació passen per alt.

Aquí és on entra en joc l'aplicació que vull crear. Una aplicació web on poder ajuntar el màxim de coses que he trobat a faltar en aquestes dues empreses, i no pel fet que no féssim servir res del que crearé, sinó perquè com he dit, mai he vist que es donés una importància suficientment alta en coses bàsiques com un simple calendari d'esdeveniments on gestionar les reunions de l'empresa, un servei de missatgeria "centralitzat" en una mateixa aplicació per comunicar-te amb l'equip o un simple mur de publicacions on comentar què s'està fent o alguna notícia que pugui ser útil a l'equip.

1.2. Món *Startup*

Com comentava al punt anterior, el món de les *Startups* ha evolucionat espectacularment en aquests últims anys. Com explica aquest article de La Vanguardia, només Barcelona i Madrid van tancar el 2018 amb uns fons d'inversió de 1.340 milions d'euros (un 70% més que el 2017) i un total de 4.100 empreses (un 30% més, en aquest cas, que el 2017). A part, Barcelona és la 5a ciutat europea pel que fa al rànquing d'inversions, amb un total de 871 milions d'euros, i la situa en una de les principals ciutats predilectes a l'hora d'apostar per l'inici d'un negoci dins l'àmbit tecnològic.

També és cert que el canvi més gran en aquest darrer any s'ha produït a nivell d'inversió i no pas tant a nivell de creixement d'empreses. Això, segons explica l'article del Periódico, és degut al fet que les empreses existents estan evolucionant molt favorablement i donen a veure un símptoma de maduresa gràcies a la duplicació del capital que han generat en aquests últims 12 mesos.

És per aquesta raó, juntament amb la meva involucració a nivell professional, que he decidit centrar el projecte en aquest món. Veig que hi ha moltes oportunitats diferents a l'hora de crear equips, projectes, idees... I per la mateixa raó, també hi ha oportunitats per tal de poder desenvolupar eines que afavoreixin el treball i l'evolució interna en cada una d'aquestes petites empreses que comencen arrel d'una idea. Òbviament, el ventall és molt ampli per poder crear un producte que pugui servir per a tota classe d'empreses, però generar una primera idea per poder rebre el màxim de feedback possible i així veure cap a quin sector o tipus d'*Startup* pot encaixar més, és un primer pas força important.

Si fes un anàlisis pel que fa a la meva experiència professional respecte tot aquest tema del creixement i de les oportunitats que hi ha cada cop més a l'abast de tothom, el resultat seria totalment positiu. M'explico.

A priori pot semblar que el fet de treballar a una *Startup* no és una bona manera d'entrar al món laboral pel simple fet que no és una empresa suficientment gran i no et podrà donar les mateixes condicions que una altra que ja porti anys al sector i un producte conegut i exitós. És cert que com és d'esperar, una *Startup* no té el mateix nivell econòmic que una consultora, per exemple, però personalment penso que no és algo que s'hagi de buscar, sobretot quan entres per primer cop en aquest món i t'interessa aprendre i formar-te.

Arran d'aquesta última reflexió i seguint amb el tema de per què una *Startup* pot sembla no ser la millor de les primeres opcions, jo vaig decidir tornar a entrar en una al canviar de feina a causa de la bona experiència que havia tingut amb la primera. És clar que no tot és meravellós i molt probablement hi haurà més feina i responsabilitats que a una empresa normal, degut al reduït nombre de treballador i les "presses" que hi ha per voler treure el mercat un producte el més ràpid possible per començar a generar ingressos i així no dependre dels inversors, però això precisament va ser el que em va fer pensar en aquest projecte i en veure on realment hi podria encaixar una eina que ajudés a poder millorar les coses que jo havia viscut a les dues empreses i que veia que si s'havien repetit en ambdós casos, ho podria retrobar perfectament en una tercera.

1.3. Propòsit

Molt simple: iniciar la creació d'una aplicació web online on els treballadors de l'empresa puguin centralitzar tot coneixement i comunicació. Dic iniciar perquè al passar per les dues *Startups* en les que he treballat, tot i ser del mateix àmbit tecnològic i estar en la posició de desenvolupador web, he vist que hi ha problemes força diferents i maneres d'afrontar l'organització totalment desiguals. Per aquesta raó, és força difícil crear una aplicació on poder resoldre el 100% dels problemes organitzatius i comunicatius de totes les empreses però de la mateixa manera que empreses del nivell de Facebook o Twitter no van solucionar els problemes el primer dia del seu llançament,

em conformaria desenvolupant una primera versió per tal de veure com ha ajudat les empreses i quines millores o, fins i tot, cap a quin camí hauria de centrar-se el futur desenvolupament. També cal dir que, per què no, podria esdevenir en més d'una aplicació web diferent. Hi ha casos que per voler intentar incloure molta cosa a solucionar, el producte perd l'essència i molts cops val més pensar si cal tirar per segons quines millores o senzillament separar la funcionalitat en dues aplicacions diferents amb propòsits alternatius.

Tornant a l'experiència personal, en aquest cas respecte a les eines que he fet servir, n'hi ha de molt bones òbviament, no estem creant el nou Google. Però sí que he vist que totes fan molt bé una o dues tasques: rebre correus, generar planificacions, missatgeria... Sempre he trobat a faltar, però, una unificació de totes aquestes petites tasques. És difícil desenvolupar una cosa que faci tot lo anterior molt ben fet, ho tinc clar, però estic força convençut que agafant la petita i millor idea de cada una d'elles pot néixer una aplicació força decent.

2. Anàlisis

2.1. Abast del projecte

Com he dit, tinc present una plataforma molt senzilla de visualitzar, de navegar-hi i de poder fer les tasques diàries de la manera més còmode possible. No vull complicar-me dissenyant mil-i-una funcionalitats per tal de fer-la popular ni atractiva en molts sectors. Vull que faci les coses que ha de fer, per poques que siguin, però que les faci bé.

Una de les coses que m'agradaria implementar és el fet de crear una aplicació web que treballi sobre una *API* de la qual es pugui anar alimentat i així tenir tota la flexibilitat possible per si més endavant es vol desenvolupar una aplicació mòbil, d'aquesta manera ja tindríem molta part de lògica creada.

El fet de centralitzar la comunicació i tractament de dades en un punt en comú t'obre moltes portes a l'hora de desenvolupar mòduls i aplicacions independents entre si, i això és molt positiu de cara a qualsevol client, ja que li estàs oferint uns serveis però no li estàs tancant les portes directament a res.

Per tant, resumint, l'abast inicial del projecte quedaria dividit de la següent manera:

- Creació d'una base de dades amb *MongoDB* per guardar totes les dades dels usuaris d'una manera escalable i ràpida.
- Creació d'una *API RESTFul* per poder gestionar tot el tractament de les dades i la lògica que hi aplicaré. També serà l'encarregada de controlar la seguretat d'accés i el flux de connexions. Aquí hi recau una part crítica pel que fa a la importància que té una aplicació d'aquestes característiques "oberta" al públic, la seguretat ha de ser robusta i actualitzada cada cert temps per mantenir les llibreries al dia.
- Per últim, l'aplicació web seria el primer desenvolupament que faria com a eina de gestió per Startups que vull crear. Per què dic primer desenvolupament? Perquè com ja he explicat, la *API* dóna molta flexibilitat i no descarto poder crear una futura aplicació mòbil utilitzant tot el que he fet per la web.

2.2. Objectius

Els objectius principals d'aquest projecte recauen totalment a l'aplicació web. A partir de llavors, es dissenyarà la lògica associada i els models de la base de dades. Però comencem mostrant què vull crear i exactament, on arribarem.

L'aplicació web tindrà les següents funcionalitats:

- Els usuaris han de poder penjar publicacions a un taulell comú al qual tothom hi tindrà accés. Crec i confio molt en la comunicació de tot l'equip, en mostrar interès per qualsevol tema ja sigui o no de l'àmbit laboral, per tal d'estar informats i que tothom es vegi integrat a l'empresa.
- També tindrà una secció d'esdeveniments per tal de gestionar tot el que són reunions, dinars, sopars, *meetups*... Tenir centralitzat un punt comú on tothom sap quins esdeveniments se celebren, on i quan, permet que sigui molt més fàcil de coordinar l'equip. A part, també m'agradaria poder dissenyar un calendari que aparegués només entrar a l'aplicació on es poguessin marcar els dies hi ha algun esdeveniment pendent, així a simple vista és encara més fàcil de cridar l'atenció de l'usuari.
- Part de missatgeria, indispensable. Com a tota aplicació d'empresa, a part de tenir la secció comuna on s'intercanvien publicacions entre tots els usuaris, també penso que hi ha d'haver un via per poder comunicar-te en privat amb qualsevol usuari. La secció de missatgeria permetrà tenir conversacions amb els treballadors de l'empresa privadament, de moment entre dues persones.
- També m'agradaria incorporar un buscador de treballadors una mica més elaborat que no pas la barra típica superior on només surt el nom i la foto. M'agradaria dedicar una secció per poder cercar a qualsevol persona i tenir una visió ràpida de la seva posició a l'empresa, la data que va entrar i quantes publicacions a compartit, per exemple. És una manera encara més ràpida de poder saber un mínim detall d'informació per no haver de llegir tot el perfil sencer d'una persona.

- D'altra banda, com és obvi, també tindrem la secció per visitar el perfil complet d'un usuari, on veure les publicacions que ha fet, els esdeveniments que ha organitzat i tota la seva informació bàsica.
- Per acabar, la part d'administració de l'aplicació, a la qual només hi podran accedir els usuaris amb un cert nivell, permetrà crear nous usuaris juntament amb la generació de *tokens* de registre únics per aquests. Això permet un registre "privat" que només coneixeran les persones que hagin d'entrar a l'empresa. Al final és un altre mètode de seguretat per evitar que qualsevol persona tingui accés d'entrada.

Aquests serien els punts principals i obligatoris que hauria de complir l'aplicació de base. Això sí, tots aquests punts són perfectament ampliables i haurem de veure quins són els requeriments que no compleix aquesta base que s'hi hauran de sumar per tenir una estructura clara del que he de dissenyar i presentar com a producte final.

2.3. Requeriments

Donats els objectius principals del desenvolupament del projecte, ara toca avaluar els requeriments del client per tal de treure una planificació final sobre la qual començar a treballar.

Les indicacions del client respecte al projecte fan referència principalment a la part de la web, on al final és on es veu el resultat de com ha quedat tot i quina aparença presenta davant de l'usuari. Tot i així, també hi ha alguns punts a comentar pel que fa a la base de dades i la *API*. Per deixar-ho el més clar possible, separaré els requeriments en els 3 punts principals del projecte:

- DB:
    - Pel que fa a la base de dades, el client no ha puntualitzat cap arquitectura en concret. Simplement, el punt haig de tenir més clar a l'hora de construir la principal estructura és que ha de ser escalable i mantenir un trànsit de dades més que suficient que serveixi tant per a una *Startup*

petita (de 5 a 50) com per a una que ja sigui consolidada amb un cert nombre de treballadors (de 50 a 200).

  - Com he comentat, el fet d'utilitzar MongoDB ens cobreix perfectament aquest punt, ja que l'escalabilitat i el fet de treballar amb documents està molt focalitzat en aquest tipus d'aplicacions.

- API:
  - Amb el tema de la *API* no ha comentat res, però aquí m'hi he avançat jo. En comentar el tema de l'escalabilitat a la part de la base de dades, la *API* és un part molt important a l'hora de gestionar les dades i tot el tractament que aquestes reben abans de ser enviades a la web o retornades a la base de dades. Per tant, és un punt clau que hem de tenir molt en compte.
  - De la mateixa manera, com també he explicat abans, una *API* permet tenir una gran flexibilitat a l'hora d'expandir el producte cap a altres plataformes i obrir nous camins de desenvolupament.
- WEB:
  - Finalment, la part de l'aplicació web. Aquí és on s'ha demanat la gran part dels requisits mínims a complir.
  - Per una part, el client voldria que l'aplicació fos molt senzilla a simple vista, un entorn molt *user-friendly* que sense tenir coneixement avançat del seu funcionament es pogués utilitzar sense cap problema. Això pot semblar a priori un requeriment molt simple d'aconseguir, però el disseny d'una aplicació de les nostres característiques és una de les parts més complicades, ja que no s'ha de dissenyar només pel que es vol construir en aquesta primera fase, sinó que també s'ha de tenir en ment un disseny que evolucioni de la mà de tots els canvis futurs que s'hi facin.
  - Seguidament, ens ha proposat un seguit de colors per a la part gràfica de la interfície que van de la mà amb tot el *branding* de l'empresa. Per tal de fer-ho personalitzat, haurem de dissenyar una primera versió seguint aquests mínims estàndards donats, amb la possibilitat de poder jugar-hi

o deixar-ho a elecció del client en qüestió que utilitzi l'aplicació en un futur.

- Pel que fa a la part dels esdeveniments, al client li agradaria poder incorporar Google Maps per la gestió de creació i visualització d'aquests. Segons ha comentat, és una eina molt utilitzada i fa que l'usuari final interpreti més fàcilment els llocs i hi sàpiga navegar.
- Finalment, pel que fa a l'aplicació en general i segons el que ha vist en altres xarxes i que ha trobat a faltar, li agradaria poder incorporar la comunicació a temps real entre la API i la part visual web. Això ha deixat clar que no era un requeriment indispensable com a tal, però que valoraria molt favorablement si es pogués aconseguir. Considera que és una aplicació que no ha d'estar activa tot el dia, que és més un lloc on consultar-hi coses puntualment durant el dia i molts usuaris la poden deixar oberta per anar-hi fent visites de tant en tant. A nivell d'experiència d'usuari, tornar a una web al cap de 2h i no veure les últimes publicacions o no rebre els missatges mentre estem fent una altra feina no és gaire agradable. Al contrari, fa que per molt ben dissenyada l'aplicació acabi perdent l'interès de l'usuari. Per això valora tant que puguem anar rebent les publicacions, missatges o esdeveniments que es van creant al mateix moment i així sentir que estan passant coses i la gent hi participa activament.

Aquests han estat tots els punts comentats durant la reunió prèvia a l'inici del desenvolupament de l'aplicació, i com he anat explicant, els he entès i comparat favorablement respecte als objectius que havia plantejat a l'inici.

2.4. Possibles obstacles

Un cop valorats els requeriments, hi ha dues coses que haig de tenir en compte a l'hora de desenvolupar l'aplicació:

- La primera és el temps que em portarà desenvolupar a mi tot el projecte establert. Aquest punt el comentaré a continuació, però s'ha de poder fer una

estimació del temps mínim dins del qual l'aplicació hauria d'estar acabada, els recursos econòmics dels quals disposem i quanta gent hauria de participar-hi per tal de complir tots els punts d'entrega correctament.

- Per altra banda, la part de l'aplicació a temps real. Com el client ha comentat, no és un requeriment indispensable, però m'agradaria que fos possible incorporar-ho dins del desenvolupament. Això (per aquest motiu ho explico a la secció de possibles obstacles) pot comportar una lleugera càrrega de treball a sumar a tota l'organització feta prèvia a la reunió amb el client, Òbviament, moltes de les altres coses també són canvis que s'hauran de tenir en compte, però aquesta en especial té un valor afegit.
- El fet de desenvolupar una aplicació amb gestió de dades a temps real implica feina a les 3 parts del projecte. Per una banda, hem de gestionar els usuaris connectats a cada moment, amb un identificador únic per cada un d'ells que ens permeti saber qui són i a on enviar la informació que els pertoqui. Després tenim la *API*, la qual haurà de saber quines dades ha d'enviar i a qui les ha d'enviar (a part de tota la feina bàsica de guardar i retornar dades). Finalment l'aplicació web, la qual haurà de gestionar l'entrada de dades en qualsevol moment, poder-les quadrar amb l'estat actual d'aquell moment a l'aplicació i poder actualitzar-se en segon pla quan el client no hi estigui treballant. En resum, és digne d'un bloc sencer de feina per a realitzar en un sol *Sprint* (o dos), però com he dit, ho intentaré assumir.

2.5. Plantejament del projecte

De la mateixa manera que he estructurat la feina als objectius i ho faré a cada un dels *Sprints* que crearé, he decidit plantejar-me la manera de desenvolupar cada un dels 3 blocs que formen el projecte.

Primer dissenyaré cada secció per separat, començant sempre per la base de dades, on crearé la estructura de dades que ha de contenir aquesta part del projecte i generaré dades de manera automàtica per tenir un primer plantejament de com seria la situació en aquest bloc.

Després vindria la connexió amb la *API*, a la qual hi afegiríem les primeres rutes per consultar aquestes dades recent creades i així començar a poder fer les operacions bàsiques de lectura, escriptura, modificació i eliminació. Aquests passos seran comuns per a **totes** les seccions del projecte.

Finalment, un cop tinguem resolta la comunicació amb la base de dades i el tractament de totes les dades consultades i llestes per enviar, dissenyaré les vistes de l'aplicació web a les quals es veurà reflectida tota aquesta informació i des d'on podrem realitzar totes les accions que hem preparat a la *API* al pas anterior.

Aquests 3 passos seran els que aniré utilitzant al llarg del desenvolupament. Això em permet mantenir un nivell de productivitat equitatiu entre els 3 blocs del projecte i al mateix temps, poder testejar cada secció per separat, des de la base de dades fins a l'aplicació web.

3. Gestió del projecte

3.1. Recursos Econòmics

Segons els càlculs estimats a nivell d'hores dedicades a poder crear un projecte d'aquestes magnituds, he optat per plantejar dos resultats. Primerament desglossaré el total d'hores per poder veure millor quina quantitat d'aquests ocupa cada bloc:

- El projecte s'ha de completar en un període màxim de 5 mesos, això implica:
    - Dies laborals: 22 (mes) x 5 = 110 dies.
    - Horari de mitja jornada: 110 dies x 4h = 440h.
    - Hores per tasca:
        - Base de Dades (10%): 44h.
        - API (30%): 132h.
        - Aplicació Web (60%): 264h.

Depenent de com vulguem plantejar l'equip que dissenyaria el treball, podem separar-ho en dues opcions:

A. Desenvolupament realitzar per un enginyer **Full Stack**:
    - Aquesta persona s'encarregaria de fer el procés sencer, la qual cosa implicaria que treballaria les 440h.
    - El sou mig d'un enginyer Full Stack és de 44.000€ bruts a l'any [14].
    - Calculem el sou net per hora que cobraria:
        - Sou net anual: 44.000€ – (20.8%) IRPF – (6.35%) Quotes SS = 32.054€ nets/any.
        - Sou net mensual: 32.054€ / 12 mesos = 2.671€/mes.
        - Sou net per hora: 2.671€ / 160h mensuals = 16.7€/h.
    - Per tant, si ha de fer 440h, el sou a pagar seria de ***7.348€***.

B. Desenvolupament realitzat per un enginyer especialitzat en la part del **Back-End** i un altre de **Front-End**:

a. És probable que a nivell econòmic pugi una mica més que l'anterior, però aquí el més important seria el temps total d'implementació.
b. El sou mig d'un enginyer especialitzat en **Back-End** és de 28.500€ bruts a l'any [14].
c. El sou mig d'un enginyer especialitzat en **Front-End** és de 29.836€ bruts a l'any [14].
d. Calculem primer el sou net de l'enginyer **Back-End**:
    i. Sou net anual: 28.500€ - (15.8%) IRPF – (6.35%) Quotes SS = 22.137€ nets/any.
    ii. Sou net mensual: 22.137€ / 12 mesos = 1.844€/mes.
    iii. Sou net per hora: 1.844€ / 160h mensuals = 11.5€/h.
e. Calculem el sou net de l'enginyer **Front-End**:
    i. Sou net anual: 29.836€ - (16.4%) IRPF – (6.35%) Quotes SS = 23.048€ nets/any.
    ii. Sou net mensual: 23.048€ / 12 mesos = 1.920€/mes.
    iii. Sou net per hora: 1.920€ / 160h mensuals = 12.0€/h.
f. Per tant, tenint en compte el que cobrarien cada un d'ells, calculem el total a partir de les hores que faria cada un d'ells:
    i. L'enginyer **Back-End** s'encarregaria de la part de la base de dades i de la *API*. Això suma un total de 44h (DB) + 132h (*API*) = 176h.
    ii. El sou que hauríem de pagar a l'enginyer seria de 176h x 11.5€/h = ***2024€***.
    iii. Pel que fa a l'enginyer especialitzat en **Front-End**, ell només s'encarregaria de la part de l'aplicació web. Per tant, això suma un total de 264h x 12.0€/h = ***3168€***.
    iv. Finalment, si sumem els dos sous, el total a pagar per realitzar el projecte amb dos enginyers és de ***5192€***.

Com es pot comprovar, hi ha una diferència prou significativa a l'hora de valorar els dos preus finals, i no només pel que fa al valor total d'euros a pagar, sinó perquè el sou inferior també implica que el treball es realitzarà en menys hores. Al ser un plantejament

totalment positiu en tots els aspectes, crec que la decisió queda totalment establerta en aquesta segona opció B de ***5192€***.

Per altra banda, també hauré de planificar el cost total que suposarà tot el programari utilitzat i el desplegament del projecte a un servidor.

Pel que fa als programes utilitzats per a desenvolupar el codi, he utilitzat els següents:

- Studio 3T: Software per gestionar de la base de dades.
    - Aquest programa té un cost gratuït si s'utilitza dins de l'educació i no per a obtenir béns econòmics.
    - En aquest cas, la llicència seria de $149 (133€) per usuari.
    - A part d'aquest software, la companyia creadora (3T Software) ha adquirit un programa anomenat prèviament RoboMongo, el qual era software molt semblant a Studio 3T però gratuït. Al adquirir-lo, l'hi ha canviat el nom a Robo 3T i el segueixen oferint com a alternativa gratuïta a Studio 3T. Per tant, seria una alternativa gratuïta a tenir en compte.
- Visual Studio Code: Software per a escriure tot el codi JavaScript que he desenvolupat.
    - El programa és totalment gratuït i multi-plataforma, així que per aquest cas no hi ha problema ni despesa econòmica.
- Postman: Software utilitzat per a fer les proves de totes les peticions creades amb la API.
    - De la mateixa manera que l'anterior, també és un programa multi-plataforma i gratuït.
- Finalment, qualsevol navegador és apte per a fer totes les proves necessàries per l'aplicació web.

Només quedaria saber el cost mensual del servidor per fer una estimació del que costaria mantenir tot el codi actiu i públic. Per realitzar dita comprovació, he comparat els preus que ofereixen dues de les principals entitats que subministren aquests serveis: Amazon (Amazon Web Services) i Google (Google Cloud).

He comparat els preus basant-me amb unes característiques a contractar que trobo que serien les adequades per al projecte que s'ha de desenvolupar:

- 4 vCPUs.
- 32 GB RAM
- La mida del disc no és gaire significativa, ja que es pot anar ampliant conforme les dades van creixent. Per ara, amb 1 TB SSD en tindrem suficient.
- Per la part d'Amazon Web Services [15], la màquina que ofereixen que més s'ajusta a les necessitats estimades és el model *m2.2xlarge*. Aquest model té un preu al mes de $358.68 (321.13€).
- En canvi, Google Cloud [15] ofereix el servei amb les mateixes característiques per un total de $448.14 (410.23€) al mes.

Per tant, davant les dues possibilitats, escollirem Amazon Web Services com a plataforma per a mantenir el nostre projecte obert al públic per un total de ***321.13€/mes.***

Així doncs, tenint en compte tots els preus calculats, ens quedaria la següent taula resum:

DESPESA	VALOR EN €
Sou total enginyer especialitzat en Back-End	***2.024€***
Sou total enginyer especialitzat en Front-End	***3.168€***
Programari	***0€ ****
Servidors	***321.13€/mes***
TOTAL	***5513,23 € *****

* Tenint en compte que utilitzem Robo 3T com a software per a treballar amb la base de dades, que és gratuït, ja que tots els altres programes també ho són.

** Aquest seria el preu total pagat a l'últim més de desenvolupament del projecte, tenint en compte que durant l'últim més pugem el codi al servidor per fer les últimes proves a l'ambient adequat. Per tant, sumarem el preu total dels sous més el primer mes de la quota d'*AWS*. **A partir de llavors, haurem d'anar sumant *321.12€* cada mes.**

3.2. Diagrama de Gantt

Per tal d'organitzar el projecte i dividir la feina en un total d'*Sprints*, he creat un diagrama de Gantt que explica molt per sobre les principals tasques a realitzar a casa *Sprint*, el temps que ocuparan i el responsable. Primerament tindríem la següent taula:

TASCA	RESPONSABLE	INICI	FI
Sprint 1 - Inicialització			
DB - Crear primeres col·leccions	Back-End	16/02/2019	28/02/2019
API - Incialitzar projecte	Back-End	16/02/2019	28/02/2019
WEB - Inicialització projecte	Front-End	16/02/2019	28/02/2019
Sprint 2 - Login i Registre			
DB - Ampliació model User	BackEnd	01/03/2019	15/03/2019
API - Creació de rutes Login / Registre	Back-End	01/03/2019	15/03/2019
WEB - Disseny i lògica de les noves vistes	Front-End	01/03/2019	15/03/2019
Sprint 3 - Usuaris i Perfil			
API - Rutes per gestionar dades d'usuaris	Back-End	16/03/2019	31/03/2019
WEB - Vista Perfil i People	Front-End	16/03/2019	31/03/2019
Sprint 4 - Dashboard			
DB - Creació model Post	Back-End	01/04/2019	15/04/2019
API - Rutes Post, Likes i Comments	Back-End	01/04/2019	15/04/2019
WEB - Vista Dashboard i Widgets.	Front-End	01/04/2019	15/04/2019
Sprint 5 - Events			
DB - Crear model Event	Back-End	16/04/2019	30/04/2019
API - Rutes Events	Back-End	16/04/2019	30/04/2019
WEB - Vista Events i Events Widget	Front-End	16/04/2019	30/04/2019
Sprint 6 - Messages			
DB - Crear model Conversation i Message	Back-End	01/05/2019	15/05/2019
API - Rutes Messages i gestió WebSockets	Back-End	01/05/2019	15/05/2019
WEB - Vistes Messages i gestió d'avisos	Front-End	01/05/2019	15/05/2019

Figura 3.1: Taula resum dels Sprints i les tasques associades.

Per altra banda tindríem el calendari dels mesos, dividits en setmanes per tenir una altra visualització de com es repartirien els *Sprint* al llarg d'aquest temps. Cada *Sprint* té un seguit de colors que fan referència a les tasques principals descrites a la figura 3.1 de la secció anterior, les quals indiquen DB, API i WEB:

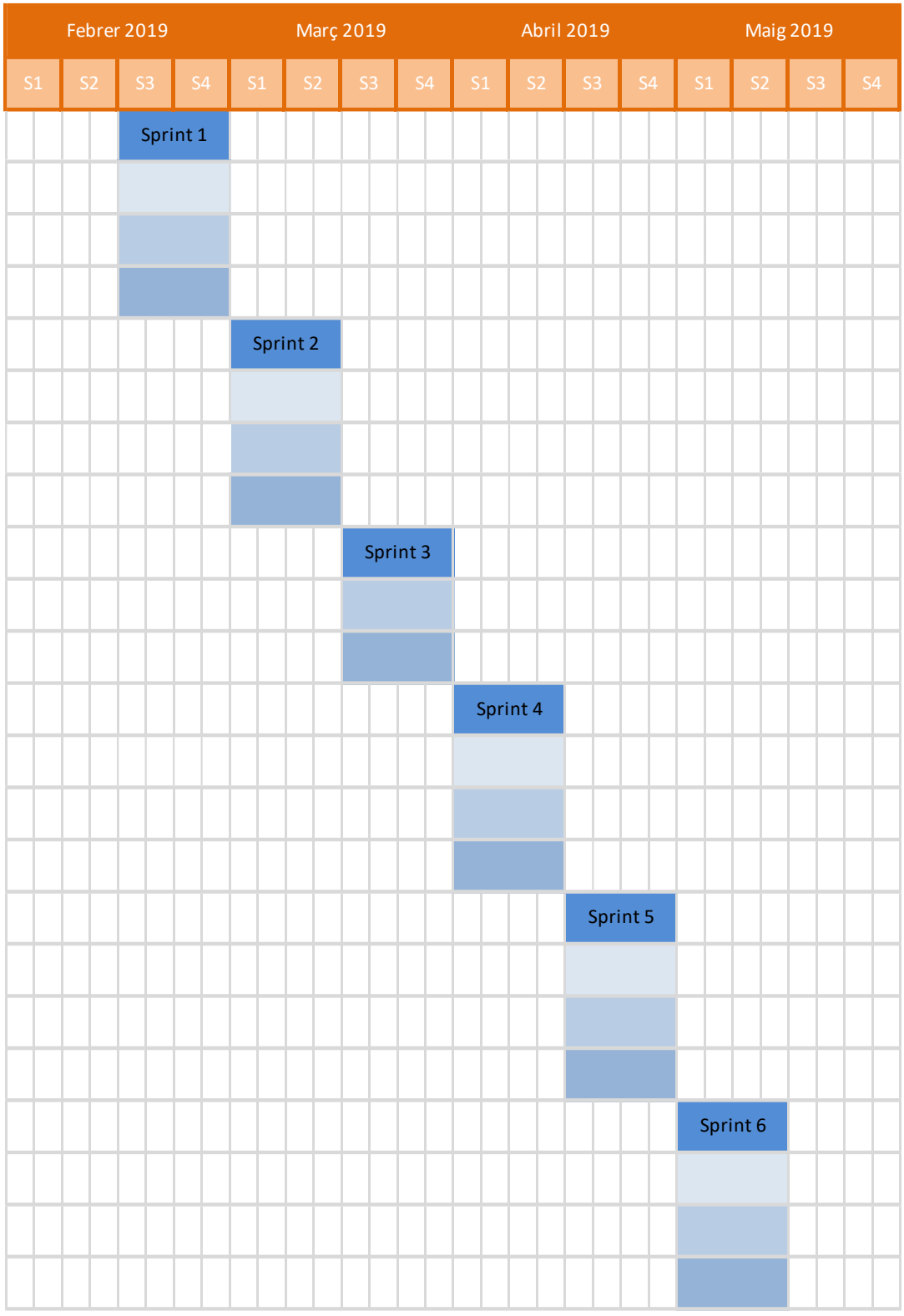


Figura 3.2: Taula on es mostra l'organització dels Sprints al llarg dels mesos juntament amb els colors de les tasques associades descrites a la figura 3.1.

4. Planificació del projecte

Per tal d'organitzar i poder gestionar les diferents tasques a realitzar al llarg de tot el projecte, he seleccionat un conjunt d'eines que he estat utilitzant els últims anys a les diferents empreses on he treballat i que m'han ajudat a agilitzar tots els processos.

4.1. SCRUM

La divisió i organització de les tasques és el punt que es podria considerar més important a l'hora de dividir un treball d'aquest abast, ja que un canvi o imprevist a meitat del desenvolupament pot fer variar els temps establerts d'entrega o finalització totalment.

És per això que he decidit fer ús de les metodologies àgils, i en concret SCRUM, per tal de planificar-me el més detalladament possible tota la feina a realitzar. Al ser un projecte de programació i creació d'una aplicació web, resulta força senzill poder separar en mòduls el conjunt de tasques a fer, de la mateixa manera que ho faríem a dins de l'equip de desenvolupadors que hi treballés. D'aquesta manera, com més individual i reduïda sigui la tasca, més ràpid es converteix el fet d'acabar tasques més grans que l'engloben, i així anar fent de la mateixa manera fins a arribar al node o tasca general. També és una manera molt optimitzada d'eliminar errors durant el procediment, ja que tota tasca té el seu plantejament reduït al mínim (crear usuari, registrar usuari, mostrar dia actual...) i resulta més fàcil recórrer el camí fins a topar amb el problema (o solució, vist així) que no s'ha implementat correctament.

El mètode SCRUM es basa en la partició de les tasques en diferents seccions d'un taulell (explicades a continuació), les quals són tractades durant els diferents períodes de temps preestablerts al començar el projecte. Això permet crear un projecte transparent a vista de tots els implicats i a l'hora facilita molt la comunicació a l'equip.

4.2. Sprints i organització

En el meu cas particular, he escollit fer períodes (anomenats Sprints) de dues setmanes, ja que ho considero temps suficient per a desenvolupar cada una de les funcionalitats per separat de l'aplicació i així tenir temps al final per encaixar els diferents mòduls i poder-hi aplicar els tests corresponents. Per altra banda, he dividit el taulell en les següents columnes:

- Product Backlog: Aquí hi apareixeran totes les tasques, sense haver d'estar simplificades al mínim (per exemple, podríem tenir el desenvolupament del login de l'usuari a l'aplicació, la qual cosa es podria dividir encara més però per ara ja ens serveix tenir la idea principal). Són les tasques principals i generals del projecte i sabem que quan no en quedi cap, hauríem d'haver acabat totalment.
- Sprint Backlog: Cada inici d'Sprint s'agafaran les tasques principals a desenvolupar i es dividiran en el màxim conjunt de funcionalitats possibles i entenedores per si soles. Aquestes es passaran a aquesta columna i són les que s'hauran de completar durant els pròxims 15 dies.
- Doing: Un cop comencem, mourem les targetes (cada una de les tasques per separat, juntament amb la seva informació i documentació associada) a aquesta columna, la qual indica que hi estem treballant i fa entendre a tota persona que si necessita alguna ajuda relacionada amb qualsevol targeta penjada aquí, ho podrà preguntar a la persona associada a dita tasca.
- Testing: Finalment, abans de donar per realitzada una funcionalitat, caldrà provar-ho amb tot el conjunt del projecte (o part, segons què estiguem creant) per tal de validar el correcte funcionament i donar-ho per finalitzat.
- Done (o columna amb el nom de l'Sprint): Aquí és on acabaran totes les tasques que hagin estat testejades i validades i que, per tant, es donen per finalitzades.

Aquesta seria l'organització bàsica del meu taulell i la que seguiré al llarg del projecte. També cal especificar un seguit de punts a tenir en compte a l'hora d'aplicar SCRUM:

- Si durant els Sprints de 15 dies no es poden acabar totes les tasques que formen la columna d'Sprint Backlog, aquestes quedaran acumulades per al pròxim

Sprint. No hi ha cap problema en fer aquests moviments, simplement ens donen un feedback respecte la organització que hem seguit a l'Sprint passat. Pot indicar que s'ha intentat incloure massa feina o que el desenvolupament ha anat més lent del que esperàvem per algun motiu en concret.

- Com he dit al principi, he dividit els Sprints segons els mòduls en els quals també he dividit l'aplicació. D'aquesta manera he assegurat que al final de cada 15 dies tindré una funcionalitat nova acabada i la podré provar amb la totalitat del projecte fins aleshores creada. El projecte està dividit amb la següent estructura:
    - *Login*
    - *Register*
    - *Profile*
    - *People*
    - *Dashboard*
    - *Messages*
    - *Events*

Per tal d'assolir aquest funcionament i poder avançar de manera eficient el projecte, utilitzaré una aplicació web anomenada *Trello*, la qual incorpora un taulell virtual per tal de poder fer més fàcil la gestió de tasques/targetes i visualització des de qualsevol dispositiu.

4.3. Git i control de versions

Durant el desenvolupament de tot projecte és molt important tenir controlar i versionat tot el codi que es genera dia rere dia, no només pel fet de tenir-ho guardat en un altre lloc diferent de les màquies locals dels treballadors, sinó perquè també és important tenir un lloc on consultar els canvis, millores, errors trobats i arreglats durant el llarg procés de creació del producte. A part, i una de les principals raons per les quals és molt útil, és pel fet de poder documentar el perquè de cada introducció de codi nou al desenvolupament, ja que si mai s'afegeix una persona nova a l'equip (partint d'una base

que el codi ja hauria d'estar documentat sempre), es podrà incorporar ràpidament llegint el seguit de canvis i els principals "per què" de tota modificació.

Dit això, he escollit GIT com a eina per a gestionar el control de versions del meu codi. A l'haver dividit el projecte en 3 grans blocs (Base de Dades, *API* i Aplicació Web) crec que dos d'aquests són idonis per pujar a un repositori i tenir un control total de tota modificació o addició de codi que s'hi realitzi.

El control de GIT és molt ampli, però al ser l'única persona que desenvolupa el projecte, intentaré basar-me en un escenari prou real per a poder-hi treballar còmodament i a l'hora tenir els grans avantatges que tindria un equip gran pel que fa al desenvolupament ordenat i coherent. És per això que he escollit el següent funcionament:

Tot projecte tindrà les següents branques:

- Branca **master**: al final de cada Sprint tot el codi validat s'haurà de pujar a aquesta branca, la qual servirà de base per passar-hi els tests periòdics i que seria on residiria el codi principal que tindríem a producció (en el cas de treballar amb un producte ja acabat i exposat al mercat).
- Branca **develop**: aquesta branca naixerà (des de **master**) i acabarà (ajuntada amb **master**) a finals de cada Sprint. S'hi realitzaran tots els desenvolupaments al llarg dels 15 dies i també podrà tenir els seus tests integrats per validar el funcionament correcte de tot el que es va creant. Si a l'acabar l'Sprint no s'han pogut finalitzar totes les tasques programades al Sprint Backlog, la branca no s'ajuntarà amb **master**. Seguirà oberta fins que pugui ser tancada coherentment, ja que ajuntar-la amb **master** sabent que hi ha alguna funcionalitat que no està 100% acabada podria ser perillós.
- Branca **hotfix**: en el cas de trobar-nos amb algun error al codi del producte que està en producció (branca **master**) i que està generant un funcionament erroni (bug), podrem crear una branca a partir de **master** amb el format **hotfix/descripció_problema** (ex: h**otfix/fix_error_when_uploading_pictures**). Aquest cas es tractarà immediatament i s'ajuntarà de nou tant amb **master** com

amb **develop** per tal de poder seguir treballant amb el problema arreglat a les dues parts.

Per tal de gestionar tots aquests canvis, branques i divisions de codi utilitzaré algunes eines que faciliten el procés:

- SourceTree: aplicació d'escriptori per gestionar fàcilment el control de versions de manera gràfica i simplificada. Permet commitejar el codi, crear branques, pujar codi als repositoris remots i veure l'arbre total de branques del projecte d'una manera molt nítida.
- VS Code [1]: a part de ser el programa principal des del qual es desenvoluparà el codi, té un terminal i un gestor de control de versions integrats. Permet veure ràpidament els canvis fets al codi i comparar-lo amb les versions antigues. D'aquesta manera es pot rectificar i tornar a un commit anterior de manera ràpida i sense haver de recuperar codi del repositori.

4.4. Repositoris

Un cop estructurat com desenvoluparem el codi, crearem els repositoris. Utilitzaré la plataforma *BitBucket*, la qual et permet crear repositoris privats de manera gratuïta i té una molt bona visualització de tots els canvis realitzats al codi. A part, fa molt fàcil totes les validacions de canvis i les **pull requests.**

Una **pull request** és l'acció que es realitza quan un branca s'uneix amb una altra (per exemple, seria el cas de **develop** unint-se a **master** al final d'un Sprint). S'hi afegeixen desenvolupadors perquè revisin el codi i donin la seva aprovació abans de fer el **merge** (ajuntar les dues branques). Un cop validat, l'encarregat responsable del codi és qui fa l'acció i el codi queda desplegat a la branca de producció: **master.**

Crearem dos repositoris, un per la *API* i un altre per l'aplicació web, cadascun amb les seves branques corresponents i el seu fitxer **README.md** associat per a tothom que vulgui saber què hi ha i com està estructurat el codi.

4.5. Divisió de les tasques

Per últim, abans de començar a programar l'aplicació, he dividit el projecte en diferents grups basant-me en els diferents mòduls que oferirà (com a mínim la primera versió) de l'aplicació web. Com he dit abans, els Sprints establerts duraran 15 dies i dividiran les següents tasques i objectius:

Sprint 1

Creació de la *REST API*, seguretat, rutes bàsiques i estructura. També es crearan les primeres col·leccions a la Base de Dades per poder fer les proves necessàries. Aquest primer Sprint generarà la gran base de tot el projecte sobre la qual anirà evolucionant. Tot i que més tard s'hi poden reforçar parts que ara no calen, és important deixar cobertes la gran majoria de necessitats per tal de crear un flux de treball lineal i no haver d'estar retocant coses a meitat de projecte que s'haurien d'haver establert de bon començament. Deixarem els tres grans blocs (Base de Dades, *API* i Aplicació Web) connectats per tal de començar.

Sprint 2

Creació de les pantalles de Registre i Login de l'aplicació web juntament amb el desenvolupament dels mòduls a la *API* i els models associats a la Base de Dades. El primer component que és necessari crear és tot el que va relacionat amb els usuaris, ja que és el punt d'accés principal a l'aplicació i el que és present a qualsevol part de la web en tot moment. Primerament ens basarem en crear dues pantalles, la d'inici (amb usuari i contrasenya) i la de registre. Aquesta última serà pública fins que el projecte no hagi avançat una mica, ja que ens interessa poder anar registrant usuaris per fer proves de la manera més ràpida possible, però la idea és que els usuaris només es puguin registrar quan se'ls faciliti una clau (o *token*) privat (des de la mateixa empresa, creat pel CEO o CTO, que són els qui tindran privilegis per a fer-ho) i a través d'aquesta clau única poder tenir accés a la vista de registre per acabar d'omplir les dades necessàries. Aquesta clau es generarà amb les dades associades a l'usuari destinatari, així que tota clau anirà lligada a una adreça de correu i un nom del treballador que l'utilitzarà, tot guardat a una taula de la base de dades des de la qual es comprovarà que una clau és

vàlida i així no es pugui forçar el registre de cap altra manera.

Sprint 3

Creació de les vistes del perfil d'usuari juntament amb les de totes les persones registrades a l'aplicació. Aquest Sprint es centrarà en el tema més proper als usuaris, creant les vistes on es mostraran els detalls de cada treballador/a i desenvolupant tot el tema de les "amistats" a dins l'aplicació. Tindrem molt present que les dades que s'obtinguin creant totes aquestes vistes "viuran" en moltes parts de l'aplicació, ja que al final s'ha d'adaptar tota la vista a l'usuari que hi està treballant i no hi pot haver cap errada amb l'edició o actualització errònia de les dades. Personalment crec que aquest serà un dels Sprints més complicats i que (esperem que no) potser pot fer allargar alguna tasca més del que seria necessari per a estar segurs del correcte funcionament del producte.

Sprint 4

Creació del *Dashboard* inicial, la finestra que veuran els clients quan entrin a l'aplicació. Aquesta vista té forces coses a implementar i és possible que la *API* creixi força aquí, sobretot amb tema de rutes i accessos a la Base de Dades. A l'haver de mostrar força informació com a punt d'entrada genèric, hauré de pensar una manera eficient de carregar les dades i guardar-les de tal manera que s'hagin de fer les mínimes crides a la *API* i es puguin reaprofitar com més millor. També s'haurà de tenir en compte el pes que pot tenir el fet de generar les publicacions en temps real (just actualitzant per a tots els usuaris al moment que un usuari en concret pengi una publicació nova) o bé mantenir un temps lògic de recàrrega que farà que l'aplicació quedi una mica més lliure de càrrega en aquest sentit. Es valorarà al començar l'Sprint i s'aplicarà el desenvolupament escollit.

Sprint 5

Creació del gestor d'events d'empresa, on es podran crear des de reunions fins a sopars informals. La qüestió d'aquesta secció és poder organitzar la màxima varietat de coses perquè no sigui (ni es vegi) que l'aplicació només està encarada al món laboral. Poder

organitzar afterworks per a les persones de l'empresa també dona un toc de personalitat a l'aplicació. Les vistes de creació i edició són el punt fonamental de l'Sprint, juntament amb la possibilitat de poder fer que els usuaris puguin seguir els events i que tothom, al mateix temps, pugui saber qui hi assistirà i qui no (dins del cercle respectiu d'amistats, òbviament). Com a punt final, depenent del volum de feina, m'agradaria poder implementar la funcionalitat que ofereix Google Maps amb la seva *API*, poden visualitzar un mapa del lloc on es realitzarà l'event i fins i tot poder escollir la localització donades unes certes suggerències mentre l'usuari va escrivint.

Sprint 6

Creació del sistema de missatgeria entre persones. Serà el sistema principal de comunicació, molt semblant al correu electrònic i que servirà per a comunicar-se entre les amistats de l'aplicació. Igual que al Sprint anterior, crear un xat on la conversa pugui ser en temps real facilitaria molt i crearia una interfície d'usuari molt agradable pel que fa a la fluïdesa de la conversació entre les dues persones. Això si, el temps de desenvolupament és superior i més complicat. La idea principal segueix sent el fet de crear un apartat de comunicació genèric, on aquesta funcionalitat del *Realtime* ajudaria però no és imprescindible. Veurem com plantegem les coses un cop arribem a aquest punt.

5. Seguiment del projecte

En aquesta secció detallaré com han estat els Sprint al llarg de tot el projecte, des de les parts tècniques més delicades i confuses a les decisions generals que s'han anat prenent durant el transcurs i evolució de tot el codi. Així mateix, dividiré aquest seguiment en els 3 blocs principals (Base de Dades, *API* i Aplicació Web) de tal manera que sigui més entenedora i compacta l'explicació a cada Sprint, ja que els 3 avancen al mateix ritme i són indispensables uns dels altres.

L'anàlisi de cada Sprint consistirà en un resum de què s'espera fer durant els pròxims 15 dies, una descripció i el resultat (resumit segons el bloc en concret que toquem) de les històries d'usuari proposades i finalment les conclusions adoptades al acabar el període, avaluant si s'ha complert els temps i tasques establertes, si s'ha pogut avançar més o menys del compte i si s'han hagut de rectificar coses que alteraran el progrés del treball. Com he explicat anteriorment, un dels avantatges de treballar amb SCRUM i els Sprints de temps reduïts és que sempre ets a temps de rectificar just al moment on veus que hi pot haver un error o que les coses s'han de tornar a avaluar per poder seguir treballant eficientment i de la manera correcte. D'una altra manera, potser seria massa tard si no féssim aquestes revisions tan seguides i, a conseqüència d'això, potser seria massa greu el plantejament que s'hauria de seguir per a rectificar l'error degut a l'avanç del projecte. Dit això, comencem.

5.1. Sprint 1

Període: 15 – 28 de febrer de 2019

5.1.1. Resum de l'Sprint

Aquest Sprint s'ha basat principalment en la inicialització de tots els projectes per tal de crear una base sòlida i coherent a partir de la qual començar a desenvolupar tot el codi. S'han instal·lat tots els paquets necessaris i creat el conjunt de repositoris per tenir un control des del primer moment (sempre seguint les especificacions dels punts 4.3 i 4.4 sobre com organitzar el codi a les branques dels repositoris). A primera vista pot semblar

un Sprint molt senzill, però el fet d'organitzar bé les coses el primer cop pot ajudar a tenir un camí planer durant el transcurs de tot el desenvolupament, ja que si després hem de rectificar coses que s'han creat a l'inici de tot, pot ser que ja no sigui tan trivial i fàcil el canvi.

5.1.2. Descripció i resultat de les històries d'usuari

5.1.2.1. Base de Dades

- DB – Instal·lar MongoDB
- DB – Crear document de configuració
- DB – Crear usuari amb permisos
- DB – Afegir model User de prova

Per aquest projecte he utilitzat la versió *4.0.3* de *MongoDB* [3]. Per defecte, si iniciem MongoDB al nostre sistema sense cap paràmetre, el servei s'iniciarà escoltant les peticions a la nostra IP local (*127.0.0.1*) i al port *27017*. L'adreça IP i el port no són cap inconvenient ara mateix, però sí que hem hagut de restringir algunes coses com ara el control d'accés a les dades (mitjançant un *user*/*password*) i la carpeta a la qual es guarden totes les dades i els *logs* (per tenir clar on buscar els fitxers necessaris si mai hi ha un error o hem d'exportar les dades a una altra màquina. Començaré explicant com hem creat l'usuari i l'arxiu de configuració personalitzat per securitzar l'accés.

Com he dit, el servei de *MongoDB* inicia a una IP i port per defecte sense credencials, la qual cosa ens serveix per poder-hi entrar per primer cop i així crear l'usuari que farem servir des de la *API* per poder-nos connectar i llegir les dades. Aquest usuari té tots els permisos de lectura, escriptura i modificació de les taules i quedarà guardat dins la taula *admin* de la Base de Dades. Fet això, toca crear l'arxiu de configuració que habilitarà el flag d'autenticació que per defecte ve desactivat a *MongoDB*. He habilitat a l'apartat *security* el parell clau/valor *authorization: enabled* i he indicat així que tota connexió s'ha d'autenticar sobre una taula (desconeguda per a tot aquell que no conegui la nostra estructura) abans de poder tenir accés a les dades.

Un cop creat l'usuari i el fitxer, només cal engegar el procés amb la següent línia:

- mongod --config [RUTA_DEL_NOSTRE_FITXER_DE_CONFIGRACIÓ]

Finalment he afegit un primer model d'Usuari, el qual em servirà per fer les primeres proves de comunicació entre els 3 blocs i que després podré aprofitar per ampliar un cop avanci més amb el desenvolupament genèric. El model consta dels següents camps inicials:

- *username*
- *name*
- *lastName*
- *email*
- *password*
- *age*
- *city*
- *mobile*
- *role*
- *joinedOn*
- *updatedOn*

Nota: Com que no estan definits totalment els camps de la taula *User* no els explicaré ara per ara, sinó que simplement ompliré la taula amb uns quants exemples per la primera fase de testing. Al punt 5 de la memòria (Implementació > Arquitectura > Base de Dades > Models) s'explicarà els models finals i cada un dels camps en detall.

5.1.2.2. API

- ❖ API – Inicialitzar projecte
- ❖ API – Instal·lació de paquets necessaris
- ❖ API – Crear servidor
- ❖ API – Crear gestió d'errors comuns
- ❖ API – Afegir seguretat *CORS*
- ❖ API – Test amb *Postman*
- ❖ API – Crear repositori

Ara que tinc el servei de la Base de Dades aixecat, necessitava una eina per poder comunicar-me entre aquest servei i l'aplicació web. Aquí és on entra en joc la *API*. Abans de res, ha estat necessari instal·lar *NodeJS* [4] a la màquina i per aquest cas utilitzaré la versió *11.6.0*.

He creat un projecte buit amb la comanda *npm init*, la qual ens pregunta en un primer moment un seguit d'opcions a omplir per tal de definir un arxiu bàsic de configuració de l'aplicació que generarà (nom del projecte, llicència, autor, dependències...). Un cop inicialitzat, he instal·lat els següents paquets bàsics que ajudaran a crear el nou servidor:

- *express (v4.16.4)* [6]: *framework* encarregat de facilitar la gestió de peticions, rutes, controladors i la lògica més potent d'un servidor amb NodeJS.
- *body-parser (v1.18.3)* [8]: *middleware* encarregat d'interpretar-nos el cos de les peticions i transformar-les a un format de tractament més simple per nosaltres, com podria ser el cas de *JSON*.
- *mongoose (v5.4.12)* [10]: eina que facilita el tractament i control de dades amb *MongoDB*. És la unió entre la nostra *API* i la Base de dades, ens permetrà fer consultes simplificades i crear models i estructures per enllaçar-los amb els que anirem creant a les nostres diferents taules.

Un cop instal·lats els paquets anteriors, ja podem iniciar el servidor. Farem que es connecti a la Base de Dades amb les credencials de l'usuari que hem creat abans i si tot va bé, es posarà a escoltar peticions al port *8000* (setejat per nosaltres). En aquest punt tenim un servidor que no té cap ruta definida i per tant no retornarà cap resposta vàlida. Abans de començar a crear les primeres rutes he establert una seguretat mínima però necessària.

La Base de Dades està protegida per usuari i contrasenya, però les peticions a la *API* poden arribar des de qualsevol adreça i tampoc no tinc una gestió d'errors creada, cosa que pot acabar produint que la *API* s'aturi o es reinici en qualsevol moment sense deixar clar quin ha estat el cas, i això no és bo. Per això, he afegit una resposta bàsica amb el codi d'error *500* per defecte i un missatge comú al final de tot del "camí de peticions", al qual arribarà la petició entrant si hi ha hagut algun problema durant el transcurs dins la *API*. Finalment he desenvolupat un control d'accés (*CORS*)

per tal de limitar la IP des de la qual podré rebre peticions i quines capçaleres acceptarem com a bones:

- Només acceptarem les peticions que ens arribin des de l'aplicació web, és a dir, del domini *http://localhost:XXXX* (caldrà substituir el port *XXXX* pel de l'aplicació web).
- Només acceptarem mètodes *GET*, *POST*, *PUT*, *PATCH* i *DELETE*.
- Només acceptarem capçaleres amb *Content-Type* (per les peticions que portin dades formatejades amb JSON o qualsevol altre format especificat) i *Authorization* (per les peticions amb *token* de validació).

Fet això, ja tinc una seguretat mínima assegurada per poder començar a treballar. Com a primer cas d'exemple, utilitzant l'aplicació *Postman*, he fet un seguit de consultes a la direcció *http://localhost:8000/user* per tal de validar que es retornin totes les dades d'exemple que he introduït a la taula d'usuaris.

A part, he afegit la ruta corresponent al fitxer *routes.js*, la lògica al controlador *user_controller.js* i un cop llest he pogut observar que totes les dades arribaven correctament, sense formatejar ni tractar, ja que ara per ara només m'interessa saber si la comunicació funciona.

Al visitar l'enllaç, tot correcte, he comprovat que el *JSON* resultant es transmet d'un extrem a l'altre. Funciona correctament!

Ara que tinc la primera versió bàsica de la *API*, he iniciat un repositori local amb *GIT* al directori corresponent i he pujat el codi a *BitBucket* per començar a tenir un control de tots els canvis que vagi generant.

5.1.2.3. Aplicació Web

- ❖ WEB – Inicialitzar projecte
- ❖ WEB – Creació de la primera vista
- ❖ WEB – Prova de connexió a la *API*
- ❖ WEB – Crear repositori

Per acabar l'Sprint, he inicialitzat el projecte de l'aplicació web gràcies a un paquet que construeix una estructura semblant a la de la *API*. Al tenir instal·lat *NodeJS*, he fet servir el seu gestor de paquets *npm* per instal·lar *create-react-app* [11]. Aquest paquet ens deixarà llest tot el necessari per començar a programar l'aplicació web sense haver de preocupar-nos per coses tan importants com la *transpilació* de *ES6* a *ES5* (per incompatibilitats de navegadors), la generació dels fitxers *JavaScript* resultants, fulles d'estils compilades... Entre d'altres moltes.

Així doncs, amb un simple *create-react-app smartnet* s'ha inicialitzat el projecte. Un cop instal·lades totes les dependències, el projecte ha iniciat sol i he pogut veure la vista per defecte que ens ha generat. Ara per ara, utilitzaré aquesta mateixa configuració base per fer la prova amb la *API*, ja que només m'interessa comprovar que tinc connexió entre els 3 mòduls que he inicialitzat dins aquest Sprint.

Un cop testejat que rebo les dades de prova de l'usuari que tinc inserit a la *DB* i que la *API* em retorna a través de l'endpoint que he generat abans, puc donar per finalitzat el primer test base dels 3 components principals del projecte.

Abans d'acabar, també he inicialitzat el repositori local amb *git init* i el remot a *BitBucket* per tenir controlat tot el codi que aniré generant a l'aplicació web.

5.1.3. Conclusions de l'Sprint

Aquest probablement ha estat un dels Sprints amb menys codi desenvolupat, ja que gran majoria de temps ha estat destinat a pensar com hauria de ser l'estructura dels 3 mòduls, quines eines utilitzaria i com ho gestionaria tot de tal manera que els 3 principals projectes poguessin conviure a l'hora i evolucionar equitativament per no perdre el ritme durant el transcurs de tot el desenvolupament. He deixat una bona base creada sobre la quan podré començar a treballar ràpidament al pròxim *Sprint* i, tot i que pot sembla força obvi, és important veure que s'han complert totes les històries d'usuari proposades a l'inici i això és bona senyal.

També m'agradaria puntualitzar que el resum de les històries ha estat un pèl més enfocat a la part tècnica, la instal·lació de dependències i l'explicació d'algunes comandes. Això es basa en el fet que m'interessa poder explicar el perquè de cada una

de les decisions que he adoptat i com les he dut a terme, ja que si en algun moment s'ha de retornar a l'inici per rectificar algun concepte, cal tenir clara la idea inicial i el perquè es va escollir desenvolupar-ho d'aquesta manera i no d'una altra.

De la mateixa manera, he preferit explicar les tasques juntes, sense separar per punts cada una d'elles, ja que eren tasques inicials molt puntuals i la forma més clara de resumir tota la feina feta era amb aquesta petita explicació que englobava totes les històries separades per projectes en un text conjunt.

5.2. Sprint 2

Període: 1 – 15 de Març de 2019

5.2.1 Resum de l'Sprint

Al llarg d'aquest *Sprint* hem creat l'entrada principal a través de la qual els usuaris accediran o es registraran a l'aplicació. El primer plantejament que vam fer va ser el de la creació de dues principals vistes, la de *Login* (on l'usuari prèviament registrat entraria les seves dades i accediria a l'aplicació) i la de *Registre* (on, primerament, seria un simple formulari que qualsevol persona podria omplir per tal de crear-se un usuari i així desenvolupar sense haver de fer el procés de registre més llarg del compte). Donats aquests dos punts d'entrada, vaig veure que seria força eficient el fet de desenvolupar la tercera vista, és a dir, un registre d'usuari final amb un *token* donat per la empresa, la que serà finalment el pas de registre que tindrà l'aplicació. Ja que estem desenvolupant unes vistes molt similars, és millor deixar-ho tot llest i haver de fer els mínims canvis ara que hi estem treballant que no un cop estiguem finalitzant un altre mòdul.

També s'han adequat altres parts de l'arquitectura de la *API* i la base de dades, la qual ha estat actualitzada per ampliar el model d'Usuari.

5.2.2. Descripció i resultat de les històries d'usuari

5.2.2.1. Base de dades

- DB – Ampliació del model User

- Al llarg d'aquest *Sprint* no s'han realitzat gaires canvis a la base de dades. L'única modificació realitzada ha estat a la taula *User*, on s'hi ha afegit un seguit de camps que s'han necessitat per a la creació de les noves vistes de l'aplicació. Els camps afegits són els següents:
    - *level*: Aquest camp determina el nivell de l'usuari dins la jerarquia de l'aplicació, és a dir, cada nivell té associat un conjunt de funcionalitats extres que l'usuari pot realitzar apart de les bàsiques i comunes. Un exemple seria el *level* 1, el qual identifica l'usuari com a *Admin* i aquest podrà crear *tokens* per a registrar futurs usuaris a l'aplicació.
    - *aboutMe*: Descripció general que l'usuari crearà per fer una petita introducció sobre qui és, a què es dedica, hobbies en general.. etc. És un camp que encara no es pot crear ni editar, però servirà un cop mostrem la informació a les vistes dels perfils d'usuari al pròxim Sprint.
    - *profileImg*: Conté el *path* relatiu de la imatge de l'usuari. Tot i que encara no es poden editar ni pujar noves imatges, aquest camp sí que és important ara mateix, ja que quan es crea un perfil d'usuari nou des de la vista de registre, s'assigna una imatge per defecte a tots els usuaris per igual fins que cada un d'ells no se l'actualitza dins del seu propi perfil. Aquest últim pas, igual que l'*aboutMe*, són tasques que es realitzaran al pròxim *Sprint*, però per ara ja tindrem la foto bàsica generada i guardada aquí dins.
    - *token*: Cadena de text generada aleatòriament per la *API* que tindrà tot usuari que hagi estat donat d'alta a l'aplicació però encara no hagi creat una contrasenya per entrar. Aquest *token* es genera a l'aplicació i és obligatori per a formalitzat qualsevol registre d'usuari. A part, aquest *token* va relacionat directament amb l'*email*, ja que d'aquesta manera en assegurarem que si una persona, pel que sigui, aconsegueix un *token*, no pugui donar-se d'alta a l'aplicació web i entrar-hi si no sap el correu amb el qual es va registrar aquest *token* per primer cop. És una doble seguretat creada per tal de securitzar una mica més aquest procés.

5.2.2.2. API

❖ **API – Creació de l'endpoint per a registrar un usuari**

- El primer de tot que he creat dins aquest *Sprint* ha estat la ruta a través de la qual podrem registrar un usuari (de forma permanent) a l'aplicació. Com he dit abans, no és una funcionalitat permanent però si que l'he aprofitat per tal de començar a adoptar les validacions a la part de la *API* que a l'hora em serviran tant per l'endpoint de *Login* com pel futur registre amb *token*. Les validacions a la part del servidor o *Back-End* són la part més crítica de totes, ja que no podem confiar "mai" amb les dades que ens envien del front o aplicació web i és aquí, a la *API*, on s'ha de fer les tasques necessàries per decidir si seguir endavant amb el procés o no.
- Els camps requerits per aquest registre són els següents:
  - *username* (obligatòri)
  - *email* (obligatòri, format correu vàlid)
  - *password* (obligatòri, mínim 3 caràcters)
  - *confirmPassword* (obligatòri, mínim 3 caràcters)
- He creat la ruta amb la validació prèvia on rebré els camps que necessito per inserir a la taula d'aquest nou usuari a crear. Per a aquesta ruta no és necessària una autenticació amb *JWT* (*JSON Web Token*), ja que l'usuari no existeix encara i no en pot tenir cap de creat per sessió. Fet això i havent validat que tenim totes les dades, com a mínim amb algun valor assignat, aquest *endpoint* està finalitzat.

❖ **API – Creació de l'endpoint per a iniciar sessió**

- Ara que ja tinc dades vàlides a la base de dades relacionades amb els usuaris, he creat l'endpoint per validar-les i així donar pas a l'aplicació. Igual que he fet anteriorment, cal verificar que rebem les dades amb algun valor i no cal comprovar si tenim un *JWT* que doni accés a aquest *endpoint*, ja que seguim sent un usuari anònim que intenta accedir al sistema i no té sessió inicialitzada.

- En aquest cas, només calen els camps *email* i *password* per tal d'accedir a l'aplicació.

❖ **API – Creació de l'endpoint per a registrar usuaris (amb Token)**

- Finalment, com havia explicat anteriorment, he creat l'endpoint encarregat de registrar usuaris amb el *token*, és a dir, de la forma que funcionarà l'aplicació un cop haguem creat tot el sistema d'usuaris.
- Aquest *endpoint* rep un *token*, un *email* i la *password* que decideix crear l'usuari. Tots els camps són obligatoris, l'*email* ha de seguir una estructura vàlida i la contrasenya ha de tenir com a mínim 3 caràcters alfanumèrics.
- Igual que les altres rutes, no cal requerir un *JWT* perquè seguim estant en un procés de registre i l'usuari no ha accedit mai a l'aplicació encara.

❖ **API – Creació de la funció al controlador d'usuaris pel registre de nous accessos**

- Ara que ja tenim la ruta i sabem quines dades rebrem, s'ha implementat la funció corresponent al controlador d'usuaris que acabarà generant aquest nou usuari. Primer de tot s'ha comprovat que arribin tots els camps requerits i un cop assegurat això, s'ha registrat l'usuari a la base de dades.
- La contrasenya ha quedat encriptada (utilitzant la llibreria *bcryptjs* [9]) ja que no volem que es visualitzi si es consulta la base de dades o algú roba l'accés. La protecció de dades (sobretot de contrasenyes) cal tenir-la molt en compte.
- S'ha comprovat que no hi hagi cap usuari amb el correu introduït per evitar la duplicació d'*email* (han de ser totalment únics).
- Finalment, s'ha afegit la ruta de l'imatge per defecte al camp *profileImg* d'aquest usuari, així un cop entri a l'aplicació en tindrà una d'assignada.

❖ **API – Crear funció al controlador d'usuari per iniciar sessió**

- Aquest ha estat un pas important, ja que a partir d'aquí es dona accés a dins l'aplicació web sencera i s'han de tenir molt en compte totes les dades rebudes. La primera comprovació que hem de realitzar és la de buscar l'usuari a la base

de dades segons l'*email* que ens han donat. Tant si no hi és com si la contrasenya desencriptada no coincideixen, retornarem error (sense indicar específicament on ha estat, si per l'*email* o per la contrasenya, així hi ha menys possibilitats de que algú pugui anar provant accessos diferents).

- Un cop hem comprovat que les credencials són correctes, el que he fet ha estat la generació d'un *JWT* amb l'*email*, l'*id* de l'usuari (generat automàticament per la base de dades quan l'hem registrat) i una clau privada guardada a la *API*, per tal de validar totes les peticions que aquest usuari pugui fer durant la pròxima hora. Així doncs, estem creant una clau que adjuntarem en totes les peticions que fem des de l'aplicació web cap a la nostra *API*, que a l'hora identificarà a l'usuari i que permetrà tancar la sessió al cap de 3h.
- Finalment, retornarem la resposta juntament amb el *JWT* perquè l'aplicació web pugui treballar amb ell.

❖ **API – Crear funció al controlador d'usuari per a registrar usuaris (amb Token)**

- Per últim, he creat la funció encarregada de gestionar el registre d'usuari un cop tinguem un *token* vàlid a la base de dades. Com que encara no hi ha cap interfície creada amb la qual generar aquests *tokens*, els usuaris els he creat amb el primer *endpoint* de registre, afegint a mà un *token* a cada usuari per tal de provar de finalitzar el registre amb aquest nou *endpoint* i funció del controlador.
- La seguretat l'he basat en la relació entre el *token* i l'*email* de l'usuari, és a dir, que si no coincideixen els dos, no és vàlid el registre. L'*email*, juntament amb el nom, cognom i rol de l'usuari, seran les dades que haurem d'introduir per generar un *token*. Per tant, en cas que algú pogués adivinar un *token*, sense l'*email* no seria un registre correcte i per tant no funcionaria.
- Un cop validada la relació, s'actualitza l'usuari amb la contrasenya introduïda al registre, encriptant-la igual que anteriorment amb la llibreria *bcryptjs* i en aquest punt ja tindríem l'usuari registrat.

5.2.2.3. Aplicació Web

❖ **WEB – Crear vista pel registre d'usuari**

- Aquestes vistes són diferents a la resta de tota l'aplicació, ja que són vistes públiques per tothom i prèvies a l'entrada a l'aplicació web. He decidit seguir un estil força semblant amb les dues, basant-me en un formulari a omplir juntament amb el logotip de l'aplicació i un text de benvinguda.
- La vista de registre bàsica ens demana un seguit de camps a introduir i al final de tot, un botó per registrar-nos, el qual no s'activarà fins que tots els camps no siguin vàlids. De la mateixa manera, quan un camp no és vàlid, un cop s'ha editat per primer cop i es passa a editar el següent, si aquest anterior és erroni, es marca en color vermell indicant que el valor introduït no és correcte.
- La validació al front permet una experiència d'usuari agradable, ja que l'usuari en tot moment sap si està fent bé les coses i així no ha d'intentar-ho vàries vegades fins que ho aconsegueixi. De totes maneres, mai es suficient en generar una validació a nivell de front, ja que pot ser alterada perfectament per tothom, fent que les dades al servidor puguin arribar alterades. És per això que he tingut molta cura en dissenyar uns controladors i rutes a la API que contemplin tots els camps a rebre i sàpiguen validar cada una de les dades necessàries per aquest registre.

❖ **WEB – Crear vista per l'inici de sessió de l'usuari**

- La vista de *Login* segueix un model molt similar a la de registre anteriorment creada. Aquí només tenim un *email* i una contrasenya a validar, juntament amb un botó que només s'habilita quan els dos valors anterior són correctes a nivell de la configuració que nosaltres haguem decidir (*email* amb estructura vàlida i contrasenya amb un mínim de 3 caràcters alfanumèrics).
- Quan la resposta de la *API* sigui que les dades introduïdes no són correctes, mai mostraren el valor erroni (en cas que només s'encertés la contrasenya, per exemple), ja que podria donar pistes de quins camps són vàlids i seguir intentant entrar a la força bruta. Sempre es mostra un missatge genèric d'error d'inici de sessió.
- També he creat un text que et permet dirigir-te a la vista de registre si encara no ho has fet abans d'intentar iniciar sessió.

Figura 5.1: Esborrany resumint com hauria de ser la vista de Login.

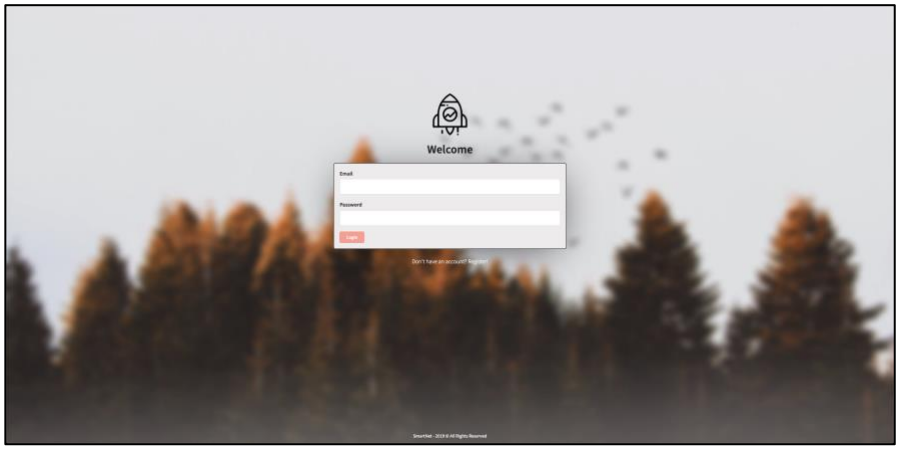


Figura 5.2: Primer versió de la vista Login implementada.

❖ **WEB – Crear vista pel registre d'usuari (amb Token)**

- Per acabar, he modificat la vista lleugerament per introduir el camp del *token* i deixar només el l'*email* i les contrasenyes (l'usuari sempre haurà d'introduir-la dues vegades, per seguretat). Faig servir la mateixa validació que anteriorment i

Figura 5.3: Esborrany de la vista de registre amb Token.

un cop el registre es completa satisfactòriament, redirigim l'usuari a la vista de *Login* perquè pugui iniciar sessió amb les noves credencials adquirides.

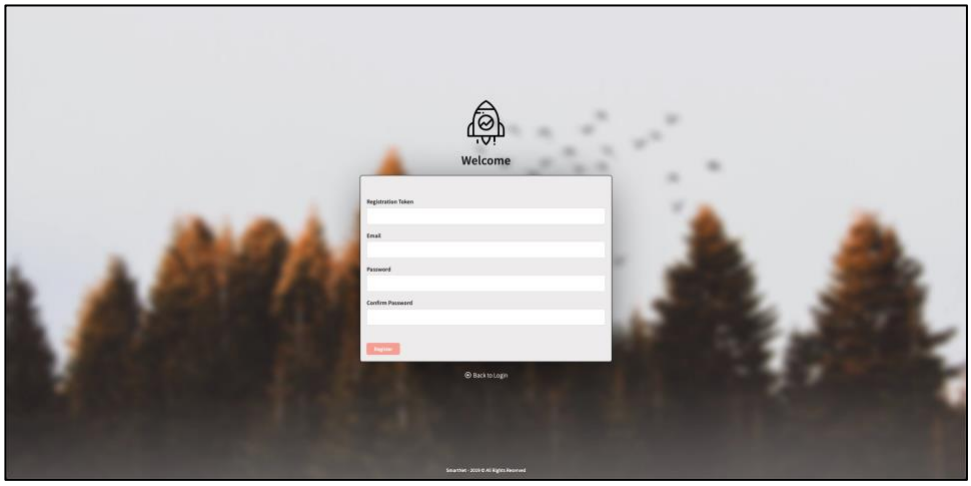


Figura 5.4: Primera versió de la vista de registre acabada.

5.2.3. Conclusions de l'Sprint

Aquest ha estat el primer *Sprint* al qual he pogut començar a dissenyar l'aplicació web com a tal, on per fi ha passat de tenir un aspecte fictici sobre uns esquemes a veure's carregant dins del navegador i ser interactiva. Sincerament, el fet de voler introduir el concepte del registre amb *token* en aquests 15 dies em va fer pensar que potser aniria una mica just de temps, però al poder reaprofitar força codi i així també validar que tan el registre bàsic com aquest nou amb *token* funcionessin, m'han fet avançar força feina que hagués hagut de fer en un futur, tornant al començament i recordant com havia fet tot el sistema de *Registre* i *Login*. Els temps han quadrat perfectament i les proves han estat satisfactòries.

Per altra banda, el fet d'haver creat unes vistes i uns *endpoints* a la *API* per registrar usuaris donats uns *tokens*, m'ha fet planificar el fet d'haver de crear una interfície a través de la qual els usuaris de nivell *Admin* hauran de poder registrar-los per primer cop i tractar tots aquests nous usuaris pendents de ser completats amb el registre i la contrasenya. Hauré de dissenyar un apartat d'administració per fer tot això i així deixar-

ho totalment automatitzat, per la qual ho faré un cop hagi deixat llest tota la part dels perfils i la cerca d'usuaris de la l'aplicació web, així el tema de tractar usuaris quedarà enllestit i podré seguir amb la creació del *Dashboard*.

5.3. Sprint 3

Període: 16 – 31 de Març de 2019

5.3.1. Resum de l'Sprint

Un cop desenvolupada la part "exterior" de l'aplicació, en aquest Sprint m'he centrat en començar a dissenyar el que serà el nucli principal, ja que ara ja tenim usuaris per entrar i validar-se dins la l'aplicació web i per tant s'han creat les primeres rutes i accessos a les diferents seccions que de mica en mica aniran apareixent.

El principal tema d'aquest *Sprint* ha estat l'usuari i tot el que té a veure amb la visualització i edició del seu perfil. Per una banda tindrem la vista detallada de tot usuari en una ruta en concret i per altra la visió general de tots els usuaris registrats, dins els quals podrem filtrar i visitar el perfil de la mateixa manera que faríem amb el nostre propi. En aquest sentit, la reutilització de vistes ha estat un punt clau que ha quedat molt transparent i compacte pel que fa a l'experiència d'usuari final.

Per acabar, s'ha implementat la part de l'administració de registres amb nivell 1 (*Admin*). D'aquesta manera, s'ha generat una secció especial on es podran veure els usuaris ja creats pendents d'activar el seu *token* (amb el registre desenvolupat al primer *Sprint*) i també la part on es podran generar aquests *tokens*, introduint les dades de l'usuari mínimes i requerides.

5.3.2. Descripció i resultat de les històries d'usuari

5.3.2.1. Base de dades

En aquest *Sprint* no he tocat els esquemes de la base de dades per a res. El fet d'actualitzar els camps de la taula d'usuari l'*Sprint* passat m'ha servit per tenir-ho tot

llest per passar directament a l'ampliació de la *API* i la creació de les vistes a l'aplicació web.

5.3.2.2. API

❖ **API – Creació d'un *endpoint* i el controlador per obtenir les dades d'usuari**

- La creació d'aquest *endpoint* és la font a partir de la qual s'obtindran les dades de qualsevol usuari de l'aplicació, sempre i quan enviem un *JTW* autenticant-nos com a usuaris registrats.
- Un cop tenim l'autenticació validada, es retornen totes les dades de l'id d'usuari enviat a la petició (sense la contrasenya, òbviament, ja que podria ser accessible des de la part del front.
- Aquesta lògica serà utilitzada tant per obtenir les dades de l'usuari quan entri per primer cop a l'aplicació com per a visitar el perfil de qualsevol usuari des de la vista de *People* a l'aplicació web. Al final, un cop seleccionat el perfil per a visitar detalladament, cal carregar les dades des de la *API*, ja que no és eficient carregar-ho tot des d'un inici.

❖ **API – Creació de l'*endpoint* i el controlador per a l'edició d'usuaris**

- Aquest *endpoint* és l'encarregat d'actualitzar les dades donat un id d'usuari. Necessita autenticació a través de *JWT* i s'utilitzarà principalment des del perfil de l'usuari que s'hagi autenticat a l'aplicació web, ja que està prohibit (lògicament) editar els perfils d'altres usuaris que no siguem nosaltres mateixos.
- Per prevenir el que comentava anteriorment, al controlador s'ha verificat que l'id usuari que envia el *JWT* per autenticar-se sigui el mateix que el que es vol actualitzar amb les dades que adjunta a la petició. Si aquesta condició no es compleix, la petició serà denegada.
- Si és correcte, actualitzarem els camps amb les dades noves rebudes i enviarem el nou usuari actualitzat com a resposta. D'aquesta manera, aprofitem una connexió a la *API* per actualitzar i a l'hora rebre les noves dades a mostrar a l'aplicació, cosa que farà que l'experiència d'usuari sigui molt positiva en rebre els canvis a l'instant.

- També cal tenir en compte que en cas de pujar una nova foto de perfil, si aquesta és diferent de la que s'assigna a l'usuari per defecte, s'eliminarà l'anterior. D'aquesta manera evitarem tenir una càrrega de fitxers al servidor, ja que tampoc és un requisit indispensable per la nostra aplicació.

❖ **API – Creació de l'endpoint i el controlador per obtenir la llista d'usuaris de l'aplicació.**

- A l'hora de cercar gent a la nostra aplicació necessitarem un *endpoint* que ens retorni tots els usuaris que hi ha registrats en aquell moment, amb les mínimes dades per mostrar una vista suficientment vàlida com per a mostrar els mínims detalls indispensables de cada usuari.
- Per defecte només carregarem els primers 40 usuaris més recents registrats a la base de dades. En cas que n'hi hagi més, caldrà fer una altra petició indicant quin volum d'usuaris més volem obtenir. Aquesta paginació s'ha creat a la part de l'aplicació web, que és al final qui sap quins usuaris té en tot moment i quins ha de demanar en cas de sol·licitud.
- Les dades que retornem són les mínimes i bàsiques, com per exemple la foto de perfil, el nom + cognom, el rol que ocupa dins l'empresa i la seva data inicial de registre, per poder veure quant de temps porten treballant a l'empresa. Com que tot document de la base de dades es retorna amb el seu identificador únic, aquest és el que està associat a cada una de les peticions que es fan quan es vol consultar un perfil més en detall, com he explicat prèviament gràcies a l'endpoint d'obtenció de dades d'un usuari en concret.

❖ **API – Creació de l'*endpoint* i controlador per registrar usuaris amb *token***

- Com he comentat, he deixat llest l'*endpoint* per poder registrar els usuaris i així obtenir el *token* que s'entregarà a cada un d'ells perquè finalitzin el procés amb la contrasenya.
- Aquest *endpoint* sí que requereix autenticació a través d'un *JWT* i espera rebre 4 camps:

- *name*: nom de l'usuari.
- *lastName*: cognom de l'usuari.
- *email*: correu electrònic de l'usuari (el mateix que haurà d'introduir al finalitzar el procés de registre i que haurà de coincidir amb el *token* generat).
- *role*: posició/càrrec que ocuparà dins l'empresa.

- Al controlador comprovo que l'id usuari associat al *JWT* correspon a un usuari amb *level* 1 com a mínim (*Admin*), ja que només aquest nivell és qui pot crear els usuaris. Un cop validat, assegurarem que l'*email* no sigui repetit i finalment generarem un *token* únic amb la llibreria *uuid/v4*.
- Retornem el *token* generat cap a l'aplicació web juntament amb la resposta per tal que puguem afegir a la llista d'usuaris pendents les dades que acabem d'inserir a la base de dades.

❖ **API – Creació de l'endpoint i controlador per obtenir els usuaris pendents de registrar**

- Un cop creat el procés per registrar els usuaris, ens interessarà poder consultar quins és el total d'usuaris que estan pendents de ser activats, amb les seves dades corresponents i el *token*, sobretot.
- He creat l'endpoint per poder obtenir aquest conjunt d'usuaris, retornant només els que tenen el camp *token* a la base de dades. Igual que l'acció de crear usuaris, aquesta de consulta també requereix autenticació a través de *JWT*.

5.3.2.3. Aplicació Web

Pel que fa a aquest Sprint, a la part de l'aplicació web he començat a dissenyar i crear les primeres vistes. A l'anar desenvolupant la *API* al mateix ritme em permet poder anar muntant l'estructura de l'aplicació més fàcilment, ja que sempre que necessito les dades a mostrar o les dades a consultar tinc l'endpoint preparat.

Així doncs, ja que ha estat el primer contacte amb l'aplicació web vista des de "dins" (un cop l'usuari s'ha acreditat amb les seves dades), he dissenyat la vista principal a partir de la qual ens podrem moure per les diferents seccions de l'aplicació web.

❖ **WEB – Crear navegació, rutes i vistes principals**

- El primer que he fet ha estat dissenyar una plantilla base la qual he dividit en un fons comú, una barra lateral de navegació amb enllaços a les seccions principals que l'aplicació ha de tenir i una barra superior amb informació rellevant d'on ens trobem en tot moment, el nostre nom d'usuari i un botó per tancar sessió.
- També he deixat creades les rutes a les diferents seccions, que tot i que encara no mostren cap contingut, ens serviran per deixar-les llestes amb els enllaços de la barra lateral. D'aquesta manera, cada cop que comenci a crear una vista nova, només haig de substituir el contingut que es mostrarà en dirigir-nos a l'apartat corresponent i tot estarà llest per a fer les proves necessàries.

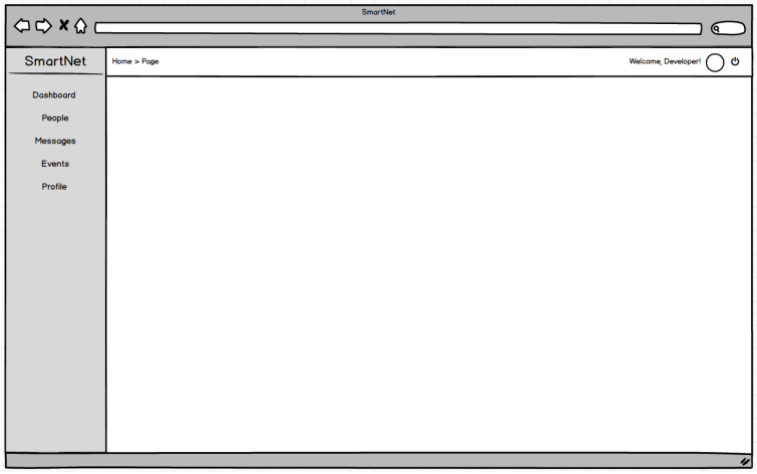


Figura 5.5: Esborrany de la primera interfície base de l'aplicació web.

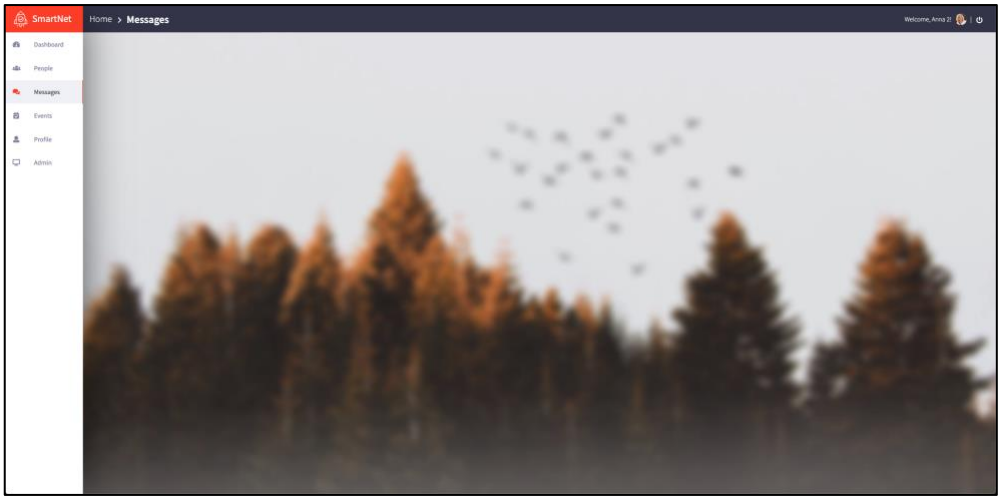


Figura 5.6: Primera versió de la interfície base de l'aplicació web acabada.

❖ **WEB – Crear vista *Profile***

- He decidit començar per la vista on es mostraran els detalls de l'usuari que estigui autenticat en aquell moment. És una vista simple on es poden consultar tots els detalls de l'usuari:
    - Dades principals
    - Interessos
    - Tot el conjunt de publicacions que hagi creat
    - Activitat recent a l'aplicació (publicacions comentades, esdeveniments als que assistirà...)
- Al ser la vista de *Profile*, també he implementat un botó que ens portarà a la vista per editar els nostres detalls d'usuari. Com és obvi, aquest botó (i la ruta que ens porta a dita vista) només estarà disponible per nosaltres com a usuaris autenticats i no es podrà editar cap altre perfil.
- Per altra banda, també he utilitzat per primer cop una nova llibreria anomenada *date-fns* [12] per mostrar la data en la qual es va registrar l'usuari. Aquesta ens permet treballar les dates que tenim guardades a la base de dades, podent-les formatejar, comparar entre elles... Serà molt útil per quan hagi de dissenyar la secció de *Posts*, *Events* i *Messages*, ja que gran part consisteix en mostrar quan s'ha creat un nou missatge o quin dia es va escriure un comentari dins d'un *Post*.

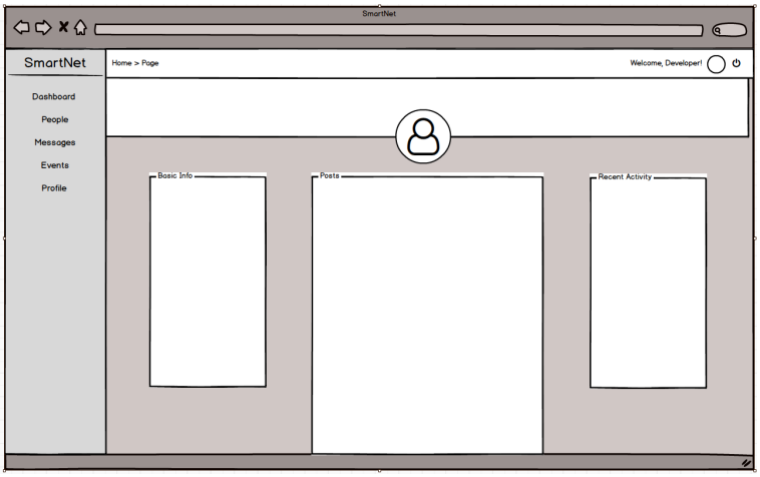


Figura 5.7: Esborrany de la vista del perfil de l'usuari.

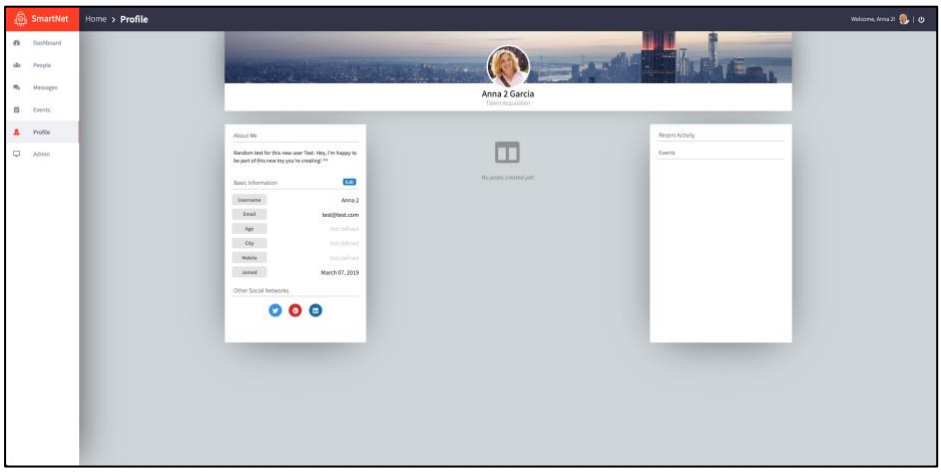


Figura 5.8: Primer versió de la vista d'un perfil d'usuari nou, sense cap publicació de moment.

❖ **WEB – Crear vista *Edit Profile***

- Seguidament doncs he completat la vista per a l'edició del nostre perfil d'usuari. Aquí hi podrem canviar les nostres dades bàsiques (cal tenir en compte que el primer cop que entrem a l'aplicació, moltes d'aquestes estaran sense completar) i també podrem canviar la foto del nostre perfil (ja que per defecte en tindrem una igual que tothom).
- De la mateixa manera que vaig securitzar la part de la *API* per assegurar-me que les dades enviades per actualitzar el perfil eren vàlides, a la part de front de l'aplicació també hi he afegit una validació. Hi ha camps (com el de l'*email*, el nom, el cognom, el rol...) que són totalment obligatoris. Altres, com el del telèfon mòbil, no és obligatori però sí que ha de tenir un format correcte en cas que el vulguem introduir.
- En cas que la validació front sigui alterada des del navegador d'un usuari, serà la *API* que refusarà la petició d'actualització.
- Finalment, un cop actualitzades les dades correctament, en guardar ens retornarà a la vista del perfil amb les dades actualitzades.
- Cal tenir en compte que les dades de l'usuari són sempre constants a tot nivell de l'aplicació i resideixen a un "repositori" central o sempre estan actualitzades. D'aquesta manera, qualsevol secció que utilitzi alguna dada també s'actualitzarà a l'instant quan editem l'usuari. Aquesta és un dels grans avantatges de treballar amb *React* [5] desenvolupant una *SPA*.

Figura 5.9: Esborrany de la vista d'edició d'un perfil d'usuari.

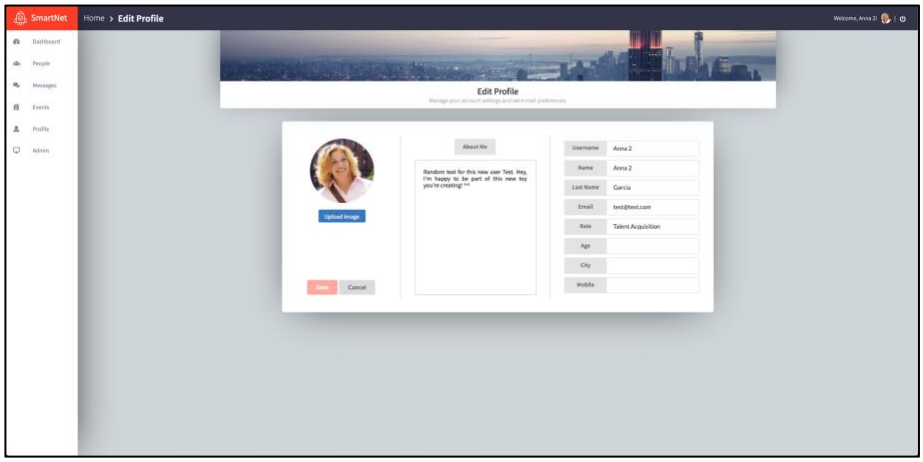


Figura 5.10: Primer versió de la vista d'edició del perfil d'un usuari.

❖ **WEB – Crear vista *People***

- Un cop definit la visualització i edició del nostre perfil d'usuari, tocava adaptar la visualització de tots els altres perfils, intentant unificar al màxim les vistes per tal de reaprofitar i així mantenir l'estil coherent de l'aplicació.
- Per mostrar tot el conjunt d'usuaris he creat una vista simplificada on apareixen tots els usuaris en forma de targetes. Aquestes tenen la informació bàsica i important per poder identificar a cada una de les persones que treballen a l'empresa (com podria ser el nom i cognom, el rol que ocupa, el nombre total de publicacions que ha realitzat i la data d'entrada a l'empresa).

- Podrem filtrar per qualsevol de les dades mencionades i tindrem a l'instant el total de targetes que coincideixen amb el filtre especificat, sense esperar a cap càrrega des del servidor, tot a l'instant.
- Finalment, al peu de cada targeta hi ha dos botons: un per veure el perfil complet de l'usuari i l'altre per enviar-li un missatge. Òbviament aquest últim encara no té cap funcionalitat, però per al primer hi ha enllaçat l'*id* d'usuari que correspon a cada una de les targetes i d'aquesta manera tenim un enllaç directe al perfil de l'usuari.
- La màgia de tot aquest procés de visualització del perfil d'un usuari X (no té per què ser el nostre com a usuari autenticat) és que cada cop que visitem la secció de *Profile* es comprova si l'*id* és igual al de l'usuari que hi ha actualment autenticat. Si ho és, es carreguen les dades que ja tenim a l'aplicació en tot moment, i si no, es fa la petició a la *API* i es demanen. Una manera molt simple de treballar i que per a l'usuari final que ho visualitza sembla totalment transparent i ràpid.

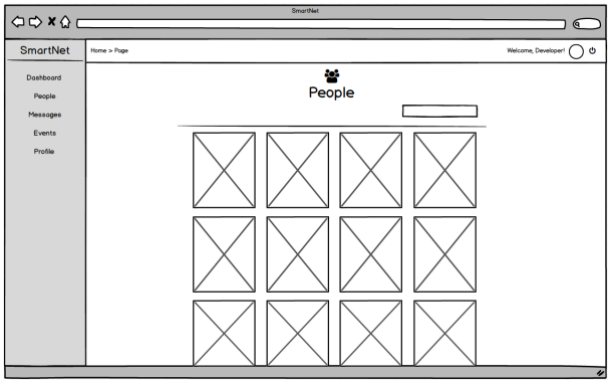


Figura 5.11: Esborrany de la vista on apareixen els usuaris de la l'aplicació web.

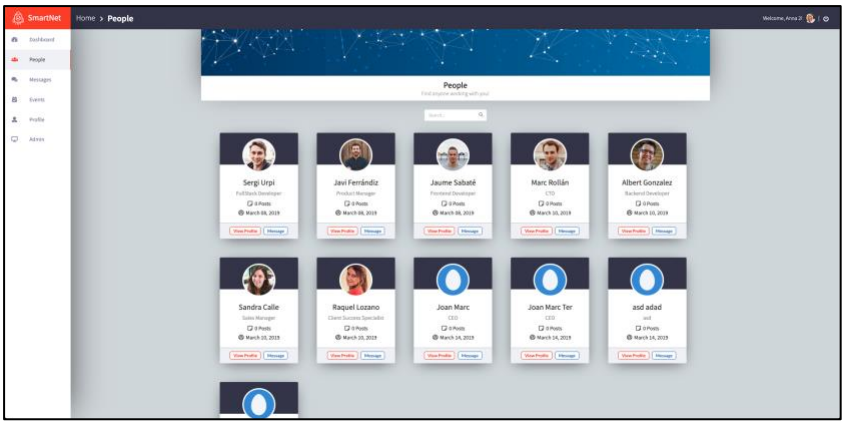


Figura 5.12: Primer versió de la vista de la secció People amb uns quants usuaris ja creats.

❖ **WEB – Crear vista Admin**

- Per acabar, i com he anat comentant al llarg d'aquest resum d'Sprint, he creat la vista d'administrador per tal de poder gestionar tot el registre d'usuaris intern un cop es dóna d'alta un futur treballador de l'empresa.
- Aquesta vista consta de dues parts: una primera on apareix una taula per poder comprovar quins usuaris hi ha pendents de registrar, amb totes les dades associades i el *token* que els permetrà completar el registre i, per altra banda, tindrem una secció on s'hi podrà crear el nou registre d'usuari (procés que generarà *token* si tot es realitza correctament).
- Dins la primera secció podrem buscar tots els usuaris pendents amb un filtre implementat que té la taula, podent buscar per qualsevol de les dades que es mostren i així obtenir al moment el que es necessiti. També podrem eliminar l'usuari en cas d'haver-nos equivocat a l'hora de crear-lo.
- Pel que fa a aquesta secció de l'aplicació web, és una vista senzilla. Però és molt útil per poder-hi anar afegint eines per a l'administració d'usuaris o de preferències en general de l'aplicació que només hi puguin tenir accés els usuaris amb *level* 1 o superior. Ara per ara, per tot el que tenim implementat és més que suficient.

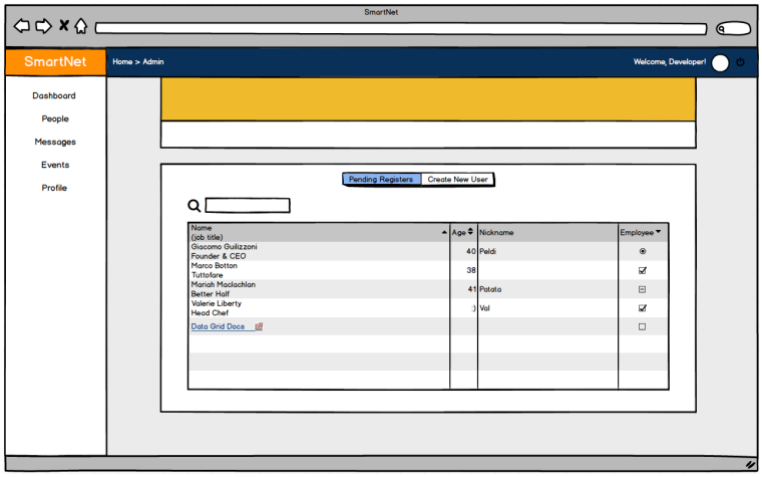


Figura 5.13: Esborrany de la vista de la secció Admin (usuaris pendents de registrar).

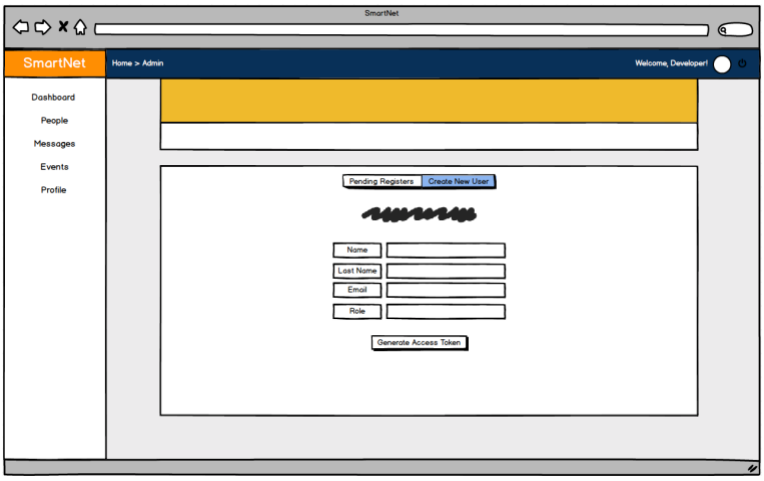


Figura 5.14: Esborrany de la segona part de la vista d'Admin (creació de Tokens).

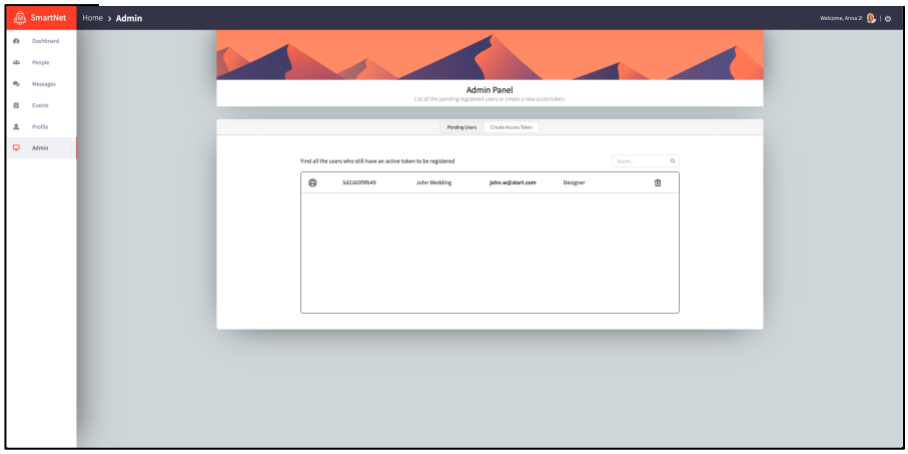


Figura 5.15: Primer versió de la vista de la secció d'Admin on veiem els usuaris pendents a registrar.

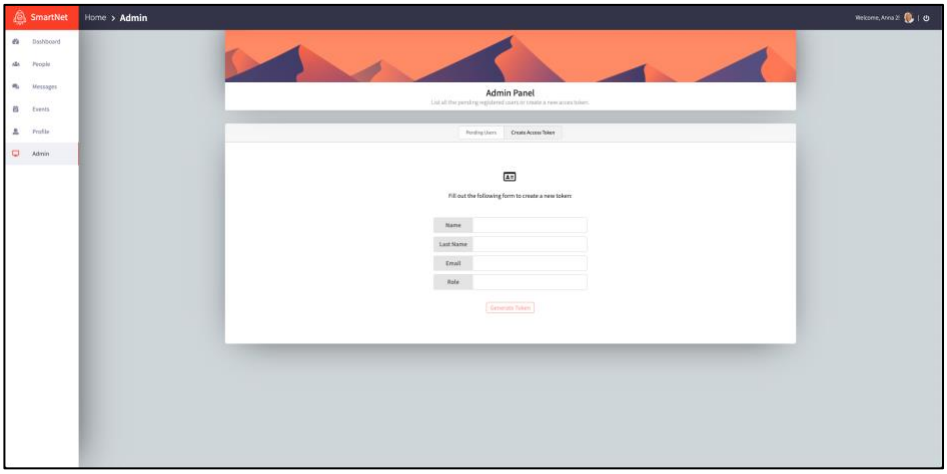


Figura 5.16: Primer versió de la segona vista de la secció d'Admin per poder crear els Tokens als usuaris.

5.3.3. Conclusions de l'Sprint

Aquest ha estat el primer Sprint en el qual he dissenyat el començament del producte com a tal a nivell d'aplicació web. Tot el sistema de rutes, seccions, control de l'usuari autenticat a través de tota l'aplicació... Moltes de les coses han començat a tenir un sentit i veure's reflectides a una interfície que cada cop va creixent més i va agafant millor forma.

El procés de creació, edició i gestió de tots els usuaris ha estat força llarg (de fet, ja vaig plantejar a la planificació d'*Sprints* que podia ser que fes allargar més el procés del que tenia pensat de bon principi) però al final he pogut gestionar bé les feines i fins i tot desenvolupar la part d'administració d'usuaris i registres amb *tokens* que no havia tingut del tot clara a l'inici.

Al tenir enquadrat el tema de les dades dels usuaris a escala general de l'aplicació hauria de facilitar molt les tasques que queden a desenvolupar, ja que al final les dades que s'han de mostrar, emmagatzemar i actualitzar depenen 100% de l'usuari que crea tota acció.

5.4. Sprint 4

Període: 1 – 15 de Abril de 2019

5.4.1. Resum de l'*Sprint*

Un dels *Sprints* més durs a nivell de feina i implementació de totes les idees que vaig plantejar de bon principi. Com ja esperava, he hagut de planificar molt bé on es guardarien les dades de tota l'aplicació relacionades amb l'usuari autenticat (en aquest cas, pel que fa a la pantalla del *Dashboard*, tot el seguit de *Posts*) per tal de poder mantenir una estructura que pogués escalar de cara a les seccions d'*Events* i missatgeria que encara queden per desenvolupar. S'ha ampliat força la *API* i l'aplicació web té una nova visió després d'implementar per fi una vista que dóna la benvinguda després de l'autenticació.

5.4.2. Descripció i resultat de les històries d'usuari

5.4.2.1. Base de dades

❖ **DB – Crear model Post**

- Per aquest *Sprint* he creat el nou model de *Posts* a la base de dades, el qual ha resultat tenir els següents camps:
  - *creator*: ID de l'usuari que ha creat el *Post*.
  - *text*: *string* que conté el missatge escrit per l'usuari.
  - *likes*: *array* amb tots els *ID*'s dels usuaris als qui els hi ha agradat el *Post*.
  - *comments*: *array* que conté objectes que descriuen cada un dels comentaris amb el text, la data de creació i l'*ID* de l'usuari que l'ha escrit.
- Com als altres models creats, també he afegit un camp de data de creació del *Post* i un altre d'actualització. Aquests dos camps es creen automàticament si afegim el *flag "timestamps"* al crear un model a la base de dades i són especialment útils.

5.4.2.2. API

❖ **API – Crear ruta i controlador per crear *Posts***

- Per començar a ampliar la *API* he creat el mètode bàsic d'afegir un nou *Post*. Com totes les altres funcions, esperem rebre un *JWT* vàlid des de l'aplicació web i també és obligatori rebre totes les dades que hem definit al model del *Post* a la DB.
- Una de les funcionalitats més esperades que tenia d'implementar era la de poder comunicar cada nou *Post* creat a tots els usuaris i que així el rebessin a temps real al seu ordinador. Això ho he implementat dins d'aquesta funció.
- He utilitzat la llibreria *Socket.IO* [13], la qual està creada sobre el protocol dels *WebSockets* i permet la connexió a temps real del servidor al client, sense que el client hagi de fer cap petició. Gràcies a això, no cal estar preguntant cada X segons i sobrecarregar el sistema. La llibreria l'utilitzaré al llarg del

desenvolupament que queda per la resta de seccions, ja que a totes hi ha d'anar una part explícita de comunicacions a temps real.

- Cada cop que es crea un nou *Post*, s'envia a tots els usuaris que estiguin connectats actualment. Ara per ara falta fer la part de lògica a l'aplicació web, però pel que fa a la *API*, guarda el *Post* a la *DB* i l'envia a temps real.

❖ **API – Crear ruta i controlador per eliminar *Posts***

- Per altra banda també he afegit l'opció d'eliminar els *Posts* que un usuari hagi creat. Esperem rebre el *JWT* des de l'aplicació web, el qual ens servirà per verificar qui és l'usuari que vol realitzar l'acció i només deixarem seguir si és el creador del *Post* que està intentant eliminar.

❖ **API – Crear ruta i controlador per obtenir *Posts***

- Un cop carreguem per primer cop la vista de *Dashboard* haurem d'obtenir el conjunt de *Posts* que l'usuari pot visualitzar. Aquí és on he implementat un conjunt de crides optimitzades a la vista per tal que a la *API* només es faci la consulta un cop a l'inici de tot, ja que la resta de peticions de nous *Posts* creats s'enviaran només quan sigui necessari.
- Així doncs, donat un *JWT*, podrem obtenir el conjunt de *Posts* més recents (ara per ara sense limitació, ja que no és una gran càrrega i, al final, sempre es pot afegir una condició a la *query*) i ordenats per ordre de creació (els més recents primer).

❖ **API – Crea ruta i controlador per afegir/eliminar *Likes***

- De la mateixa manera que enviem els *Posts* creats a temps real cap als clients connectats en aquell moment, també he afegit aquesta funcionalitat a l'afegir o eliminar un *Like* d'un *Post*.
- Quan un usuari fa *Like* a qualsevol *Post* rebrem la petició a la *API* juntament amb el seu *JWT* associat, el qual en servirà per extreure l'*ID* d'usuari i així afegir-lo a l'*array* de *Likes* d'aquell *Post*. A l'eliminar el *Like* també seguirem el mateix procés i eliminarem l'*ID* d'usuari de l'*array*. En tots dos casos, un cop hàgim actualitzat l'*array* de *Likes* del *Post* en concret, aquest *Post* s'enviarà a l'instant

cap a l'aplicació web i així es podrà veure a l'instant que hem rebut un nou *Like* sense haver d'actualitzar la pàgina ni haver de fer una altra crida a la funció anterior d'obtenir el total de *Posts* actualitzats.

- Això, a part d'evitar una càrrega de peticions al sistema, fa que l'experiència d'usuari sigui molt favorable.

❖ **API – Crear ruta i controlador per afegir comentaris**

- Per acabar amb la *API*, de la mateixa manera que he creat les funcions dels *Likes*, els comentaris funcionaran igual.
- Quan un usuari escriu un comentari, requerirem un *JWT* per validar-lo com a usuari registrat, un text del comentari i un *ID* del *Post* al qual vol afegir-lo. Un cop afegit i actualitzat l'*array* de comentaris del *Post*, l'enviarem de tornada a l'instant per així poder simular una conversa a temps real a la caixa de comentaris del *Post*.
- Com he dit abans, tant l'acció dels *Likes* com la dels comentaris ha estat una gran idea, ja que fins que no ho he programat i provat no he vist al 100% la sensació que dóna poder estar realitzant aquestes accions i veure el resultat al mateix moment.

5.4.2.3. Aplicació Web

❖ **WEB – Crear vista *Dashboard* principal**

- Aquesta ha estat la vista més difícil de dissenyar a nivell d'*UX*, ja que és la primera secció que l'usuari veu quan s'autentifica i a l'hora ha de tenir el màxim d'informació possible sense saturar. Un gran repte.
- L'esquema que vaig fer al principi per situar-me i poder tenir clar on anirien el conjunt de components que la formarien és el següent:

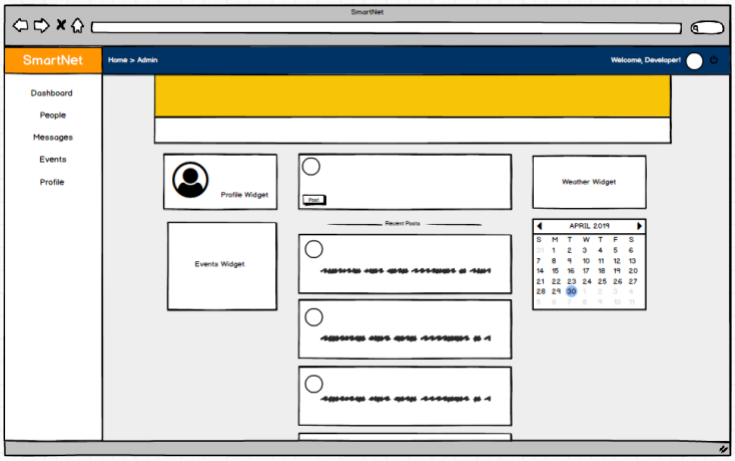


Figura 5.17: Esquema de la vista del Dashboard (juntament amb el widget dels Events).

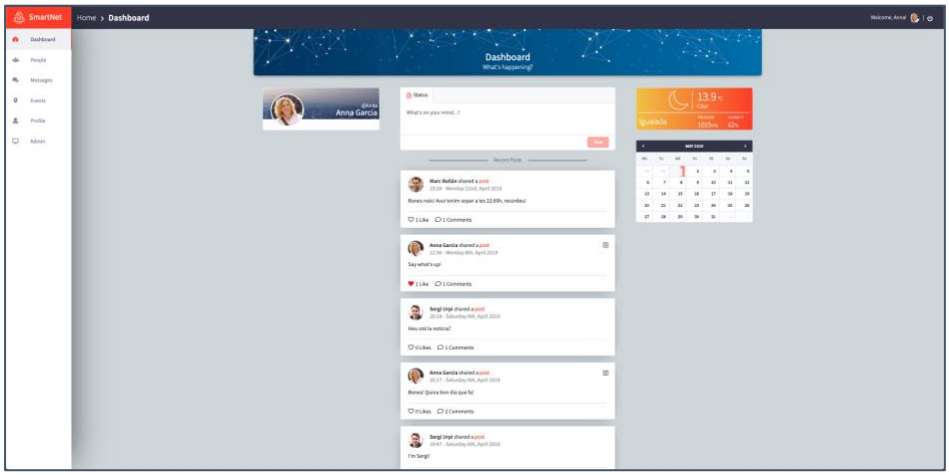


Figura 5.18: Primera versió acabada de la vista del Dashboard.

- Com es veu a la imatge, és una vista força simple a nivell visual, la qual ofereix un petit resum de conceptes que podem anar veient a diferents parts de l'aplicació com podrien ser el llistat de pròxims *Events*, la informació de l'usuari resumida, un calendari interactiu...
- Al centre hi ha la secció dels *Posts*, on primerament tenim un camp a través del qual crear un nou *Post* i sota d'aquest el llistat dels *Posts* més recents creats pels usuaris.
- Un cop ens connectem per primer cop a l'aplicació farem la crida a la *API* per tal d'obtenir els *Posts* i guardar-los al component superior anomenat *Main*. Aquest component és l'encarregat de guardar totes les dades que es van recol·lectant a la *API* i el que s'encarrega de gestionar les entrades de nova informació a través dels *WebSockets*.

- Per tal d'accedir als comentaris de qualsevol *Post*, al fer click sobre la icona que tenen al peu de cada secció individual es desplega un modal que a l'hora enfosqueix el fons i permet navegar per tots els comentaris. També té una petita secció al peu del modal on permet afegir un nou comentari.

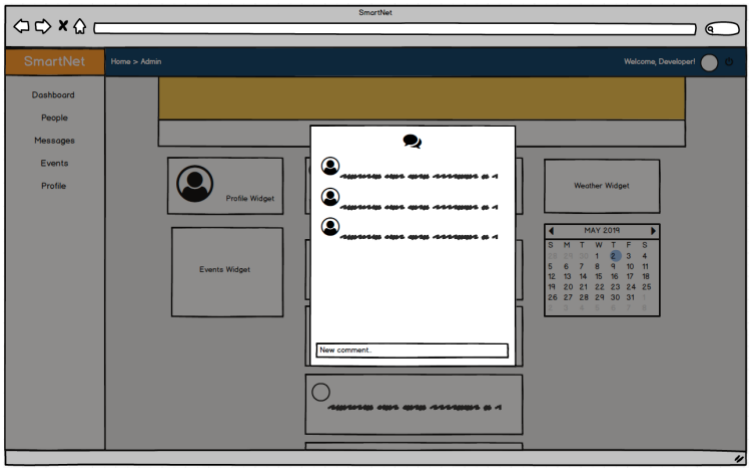


Figura 5.19: Esquema de la vista del Dashboard amb el modal dels comentaris actiu.

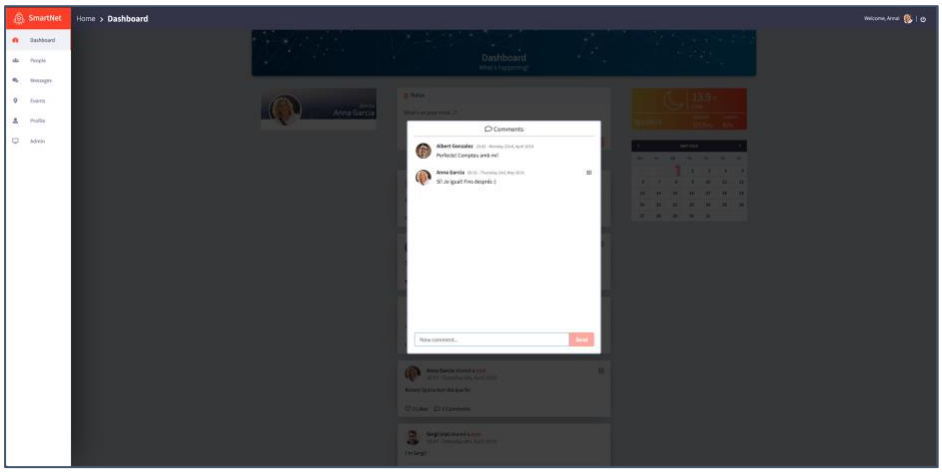


Figura 5.20: Vista de la caixa de comentaris del Dashboard finalitzada.

- Aquí també he fet servir la llibreria *Socket.IO* per connectar-me al servidor i així rebre totes les dades. En aquest cas ha estat la versió especial per la part del client.

❖ **WEB – Crear component Weather**

- El primer component (o també anomenat *widget*) que he creat aquest Sprint ha estat el del temps, el quan he nomenat com a *WeatherWidget*.

- Consta d'un petit rectangle que mostra les dades meteorològiques del lloc geogràfic des del qual l'usuari inicia sessió a l'aplicació web, com ara el temps actual (Sol, núvols, pluja...), la temperatura, humitat i pressió.
- Per tal d'obtenir aquestes dades he utilitzat una *API* pública, la qual et permet consultar amb un seguit de restriccions. Per aquesta raó, he adaptat i optimitzat la meva aplicació web perquè només faci consultes un cop cada hora (que crec que és prou sovint perquè el temps variï).

❖ **WEB – Crear component per l'usuari autenticat**

- El següent component dissenyat ha estat una petita capsa que conté la foto de perfil de l'usuari, el seu nom real i el seu nom d'usuari.
- És un simple contenidor d'informació que fa que l'usuari vegi una mica més de personalització dins del *Dashboard*.

❖ **WEB – Crear component Calendar**

- Per acabar, l'últim component que he creat a aquest *Sprint* per la vista de *Dashboard* ha estat el del calendari. Com vaig explicar al seu moment, trobo que un calendari és imprescindible per a una aplicació d'empresa com aquesta i dins del *Dashboard* encaixava perfectament.
- És totalment interactiu, permet variar entre els mesos, té un estil propi a la casella del dia actual i al passar el cursor per sobre els dies també té efectes. Es poden crear un gran seguit d'accions amb aquest calendari, però per aquesta fase inicial ho he trobat suficient.

5.4.3. Conclusions de l'Sprint

Com comentava a l'inici d'aquest resum d'Sprint, per fi he pogut implementar una de les vistes que més respecte em donava, ja que els *WebSockets* sempre havien estat una tecnologia que tenia ganes d'aprendre i al final l'he pogut posar en pràctica dissenyant aquesta part.

Per altra banda, a la foto hi apareix un 4rt component o *widget* que té el resum dels pròxims *Events*. L'he afegit a les primeres fotos o esquemes per tal de tenir una vista clara i estructurada de com haurien d'anar els altres components, però no l'implementaré fins al pròxim *Sprint*, que és quan crearé tota la part que correspon a aquesta secció.

En definitiva, ha estat la primera vista que genera forces peticions a la base de dades i això m'ha fet haver de pensar en el fet de reestructurar i crear dues coses:

- Primerament he afegit la lògica i "magatzem" de dades a un nivell superior de totes les seccions *Dashboard*, *Profile*, *Events*.., de tal manera que totes poden consultar les dades si una d'elles ja les ha demanat primer i això facilita molt el tractament i consulta de tota l'estructura de dades carregades a l'aplicació web, apart d'alliberar a la *API* de peticions i càrregues innecessàries.
- A part, com que cada cop més seccions s'han de comunicar amb la *API*, he dissenyat un servei d'accés comú que té estandarditzada la crida a la *API*. És una funció que rep un segui de paràmetres (*URL* de la petició, mètode *HTTP*, cos del missatge (opcional), *token* i opcions (també opcionals)) i d'aquesta manera tinc centralitzada tota la lògica i si mai canvia algo sé perfectament on arreglar-ho.

5.5. Sprint 5

Període: 16 – 30 de Abril de 2019

5.5.1. Resum de l'*Sprint*

L'*Sprint* en general ha estat més fàcil d'implementar que l'anterior però a l'hora també ha introduït altres "problemes", com l'ús de llibreries externes força més complexes (com la de *Google Maps*) i també ha lligat diverses seccions entre sí, com ara el *widget* d'*Events* que es mostra al *Dashboard*, els perfils d'usuari on es mostren els últims *Events* creats per cada persona... En resum, un *Sprint* força complert també.

5.5.2. Descripció i resultat de les històries d'usuari

5.5.2.1. Base de dades

❖ **DB – Crear model Event**

- Per aquesta nova secció, igual que l'anterior amb els *Posts*, he creat un model nou a la base de dades que té el format següent:
  - *name*: *string* amb el nom de l'*Event*.
  - *description*: *string* amb la descripció de l'*Event*.
  - *location*: *string* amb les coordenades de la localització on es celebrarà l'*Event*.
  - *host*: *ID* de l'usuari que ha creat l'*Event*.
  - *type*: *string* amb el tipus d'*Event* (*lunch*, *dinner*, *meeting*...)
  - *dateFrom*: data a la qual començarà l'*Event*.
  - *dateTo*: data a la qual acabarà l'*Event*.
  - *guests*: *array* amb el conjunt d'*IDs* d'usuari que assistiran a l'*Event*.
- Com es pot veure, és una estructura força simple que resumeix perfectament tot l'*Event*.

5.5.2.2. API

❖ **API – Crear ruta i controlador per crear, editar i eliminar *Events***

- Aquests tres casos són els típics de qualsevol aplicació *CRUD* (*Create Read Update Delete*) d'avui dia. Pels casos de crear i editar es demanen les dades del formulari que hi ha a la vista i per la part d'editar només cal l'*ID* de l'*Event*.
- Cal enviar un *JWT* vàlid que validarà l'usuari i així també ens permet saber si aquest *ID* d'usuari té accés per editar l'*Event* o per eliminar-lo.
- A les dues primeres accions (crear i eliminar), de la mateixa manera que es feia a la secció de *Posts*, quan un *Event* es crea o s'edita, aquest s'envia directament cap a l'aplicació web de tots els clients que estiguin connectats en aquell moment.

- Un cop un *Event* ja ha "caducat" o s'ha completat, no es pot editar més. Això ho he bloquejat tant a nivell d'aplicació (redirigint a l'usuari cap a la secció d'*Events*) com a la *API*, on retornem un error.

❖ **API – Crear ruta i controlador per obtenir *Events***

- Com a totes les altres seccions també he creat la ruta i la lògica per tal d'obtenir els *Events* el primer cop que l'usuari entra a l'aplicació.
- Necessitarem un *JWT* per obtenir la validesa d'usuari com a totes les altres rutes i retornarem tot el conjunt, ordenat per temps de creació més recent i sense límit de documents retornats de moment.

❖ **API – Crear ruta i controlador per assistir/deixar d'assistir a un *Event***

- He generat les rutes i lògica corresponent per tal de poder actualitzar les dades dels *Events*, afegint o eliminant els *IDs* d'usuari que s'uneixin o marxin d'un *Event*. És obligatori l'enviament d'un *JWT* per validar que l'usuari tingui permisos per accedir a l'*Event* en concret.
- Cada cop que s'actualitzi un *Event*, tal com he fet a les funcions d'editar, enviarem el document de tornada cap a l'aplicació web per rebre els canvis a l'instant.
- Un cop tenim el llistat d'*Events* visibles, hi ha dues maneres de confirmar o eliminar l'assistència: o bé directament fent click al botó "*Join*" sobre qualsevol targeta informativa d'un *Event* o bé desplegant la informació completa de l'*Event* al modal que apareixerà. Allà també hi haurà un botó "*Join*".
- A tots els *Events* que no siguin creats per nosaltres hi trobarem aquest botó. En canvi, als que hàgim creat amb el nostre usuari, hi apareixerà el botó d'editar l'*Event* i nosaltres hi serem implícitament com a "*Host*".
- De la mateixa manera que he fet a la part de la creació i edició dels *Events*, un usuari no pot confirmar l'assistència ni eliminar-la a un *Event* que ja hagi estat completat. Per això, tant a nivell de l'aplicació web com a la *API*, he bloquejat els accessos a aquests tipus d'*Events*. Només s'hi podrà consultar la informació.

5.5.2.3. Aplicació Web

❖ **WEB – Crear vista *Events***

- Aquesta secció s'ha pensat igual que el *Dashboard*: només es fa una crida a la *API* per tal d'obtenir els *Events* un cop. Els *Events* que es vagin creant una vegada siguem dins de l'aplicació els anirem rebent per *WebSockets* en temps real i s'aniran guardant al component *Main*, igual que els *Posts*. És un mètode que he fet servir per les seccions més grans, com ja he explicat anteriorment, i tinc intenció de continuar així, ja que proporciona una capa d'abstracció de les dades respecte la a lògica i al mateix temps millora l'eficiència de crides cap a la *API*.
- Per tal de poder filtrar entre el conjunt d'*Events* que rebem de la *API*, he afegit dos filtres. El primer és un camp de text que ens permet filtrar per nom, descripció o tipus d'*Event*. El segon és un desplegable amb el qual podrem seleccionar entre visualitzar tots els *Events*, només els que encara siguin vigents o només els que ja s'hagin completat. D'aquesta manera tenim una mica més de personalització a l'hora de navegar per aquesta secció.

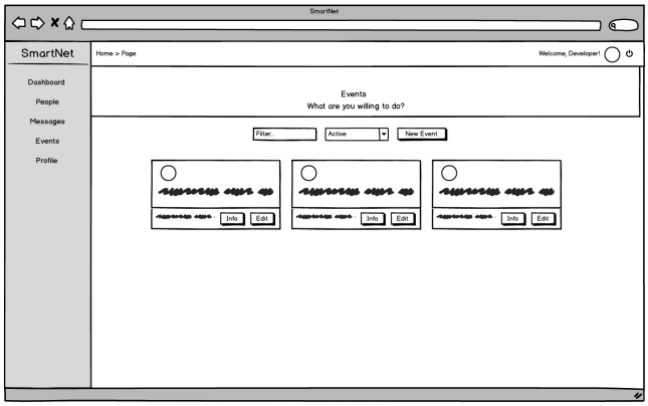


Figura 5.21: Esquema de la vista d'Events, on es veu el llistat total disponible sense filtrar.

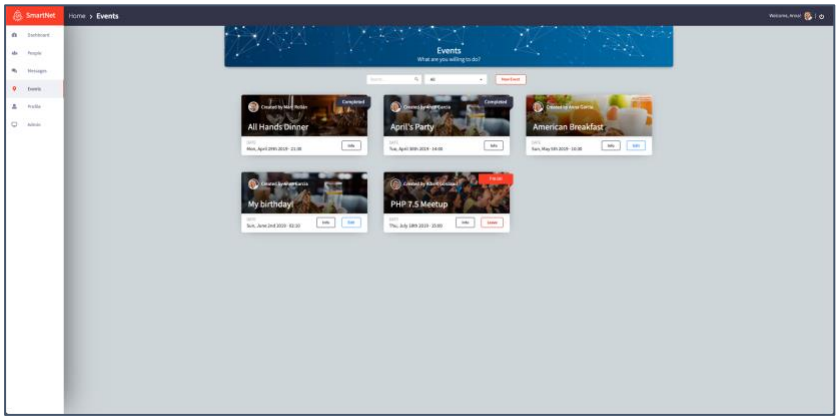


Figura 5.22: Vista de la secció d'Events acabada.

Figura 5.23: Esquema de la vista d'Events, mostrant el modal d'informació actiu d'un Event en concret.

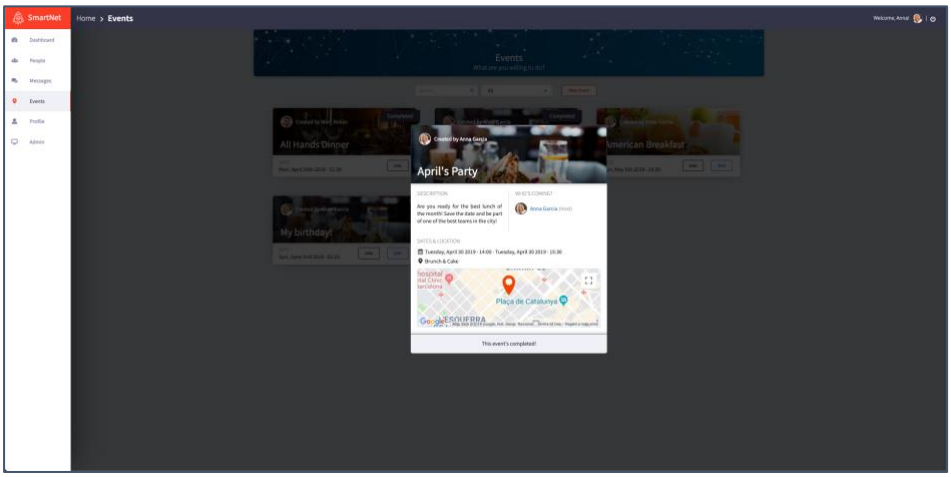


Figura 5.24: Vista de la secció d'Events acabada, mostrant el modal d'informació actiu.

❖ **WEB – Crear vista per crear i editar un *Event***

- Aquesta part l'he intentat fer força intuïtiva pel que fa a l'experiència d'usuari. He dissenyat una vista amb la qual es puguin crear els *Events* però que també serveixi per editar-los, així a l'usuari li serà més fàcil recordar què hi havia a cada camp i no ha d'estudiar més d'una vista per acabar fent el mateix.
- Tenim un seguit de camps a omplir amb text, com el nom i la descripció i a part hi ha dos desplegables: el tipus d'*Events* i la localització. Aquests dos camps ara per ara tenen unes opcions predefinides que he creat i que comprenen la gran majoria d'*Events* que es poden crear a una *Startup*. Crec que per aquesta fase inicial és suficient per a mostrar la funcionalitat d'aquesta creació i gestió d'*Events*, però, tot i que ho explicaré més endavant, una possible millora a un

futur podria ser la de permetre a l'usuari introduir una localització personalitzada o un nou tipus d'*Event*.

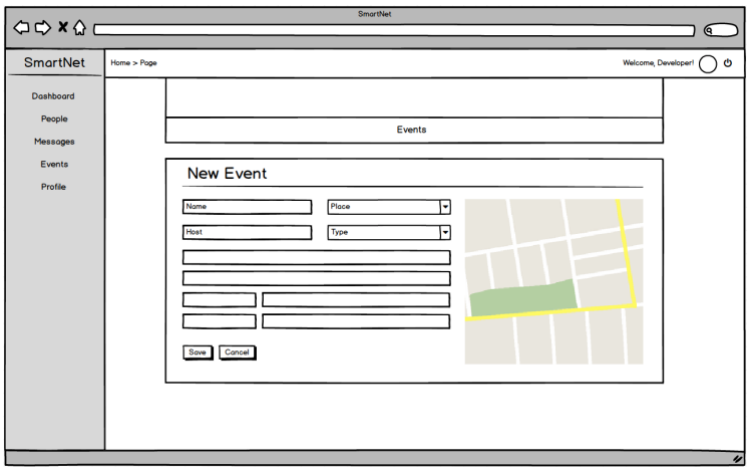


Figura 5.25: Esquema de la vista de la secció on crear o editar un Event.

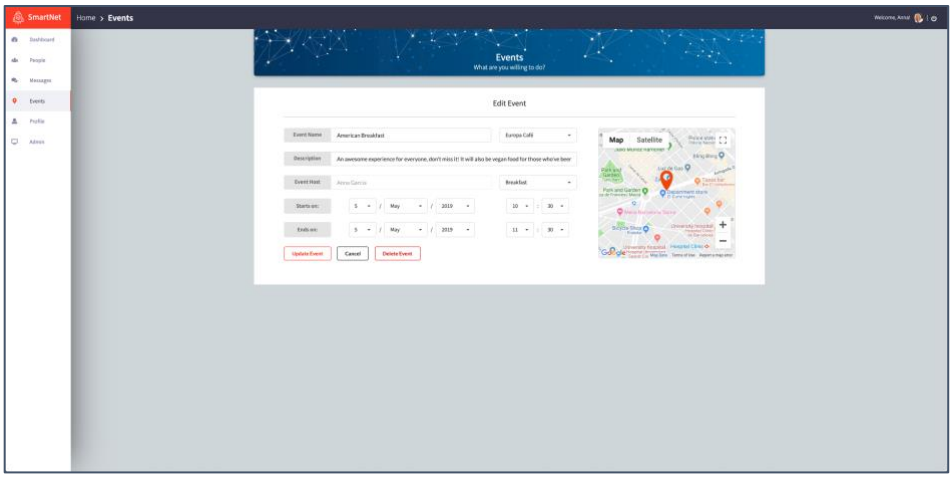


Figura 5.26: Vista finalitzada de la secció d'Events on crear i editar.

❖ **WEB – Afegir llibreria *Google Maps***

- He decidit utilitzar aquesta llibreria per fer més intuïtiu el procés de reconèixer la localització i així tenir una part "gràfica" a la creació, edició i visió general dels *Events*. Un mapa permet reconèixer una localització més ràpidament i també pot ajudar a trobar altres llocs al voltant.
- D'aquesta manera, cada cop que s'escull una localització nova, aquesta apareix marcada amb el símbol personalitzat de l'aplicació web.

- També he creat les funcions necessàries per passar les coordenades predefinides dels llocs on es poden celebrar *Events* al mapa i així saber on pintar cada marcador.

❖ **WEB – Crear component *Events* al *Dashboard***

- Aquest ha estat el 4rt component del *Dashboard* del qual parlava a l'*Sprint* anterior. Aquest *widget* o component mostra els 5 pròxims *Events* i permet accedir a la informació de cada un d'ells fent click al corresponent. Això ens porta directament a la secció d'*Events* i obrirà el modal on apareix tota la informació necessària i l'acció de poder-hi assistir.
- Com vaig dir en el seu moment, el *Dashboard* ha de tenir molta informació però que a l'hora sigui fàcil d'entendre i no saturi la vista només entrar-hi. Aquesta és una versió molt simplificada de la secció d'*Events*, que permet veure el més important dels pròxims *Events* i et dóna la possibilitat de poder-ne conèixer més simplement amb un click.

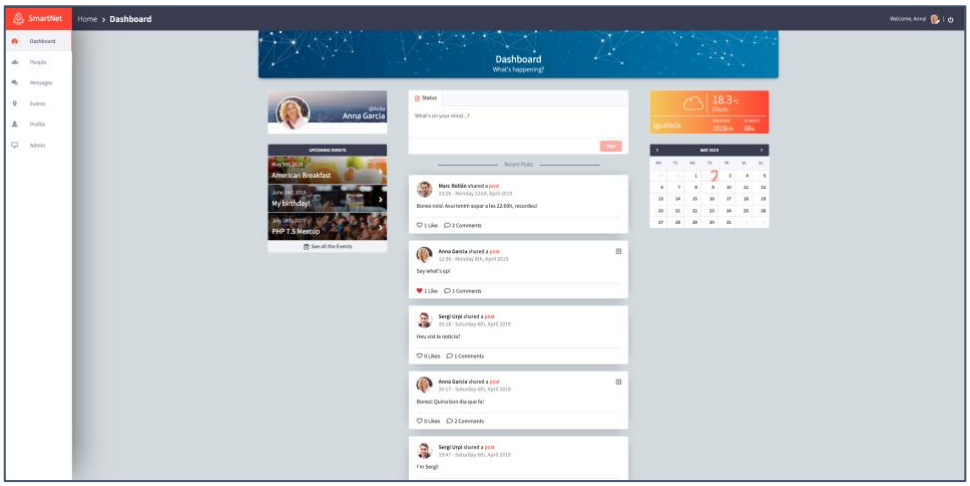


Figura 5.27: Vista del Dashboard acabada amb el widget d'Events creat.

❖ **WEB – Afegir Events al perfil dels usuaris**

- He actualitzat la secció on es pot consultar el perfil d'un usuari afegint-hi el *widget* d'*Events* corresponent a cada usuari. Així doncs, quan entrem a visitar un perfil d'usuari veurem els pròxims 5 *Events* que hagi creat. És una versió del *widget* com la que hi ha al *Dashboard* però personalitzada a nivell d'usuari.

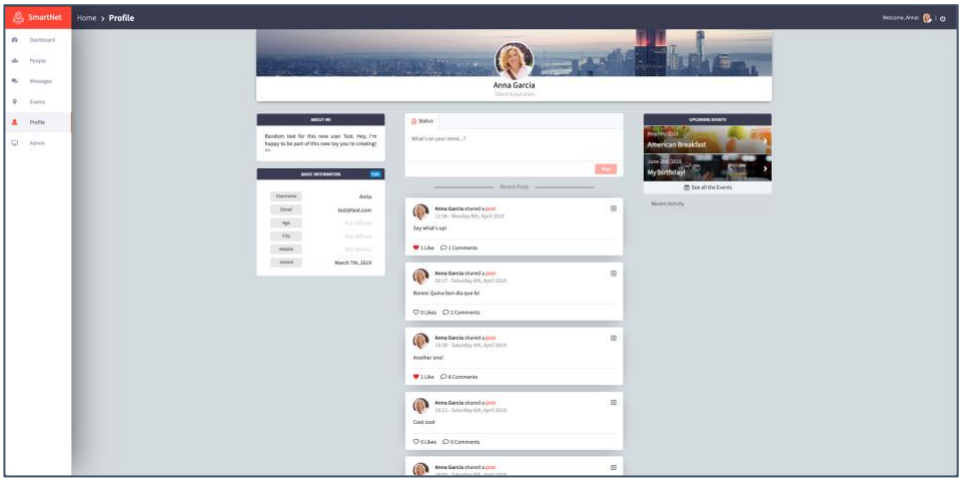


Figura 5.28: Vista del perfil d'usuari amb el widget d'Events propis.

5.5.3. Conclusions de l'Sprint

En general ha estat un *Sprint* força més entretingut que els anteriors, ja que amb la possibilitat de poder introduir la llibreria de *Google Maps* he pogut anar jugant i creant una mica més d'interacció amb l'usuari.

Pel que fa a la part dels *Events*, ha estat força semblant a la dels *Posts*, enviaments a temps real cap als clients i típiques operacions de crear i eliminar. També m'ha ajudat a entendre millor i aprendre a desenvolupar unes bones pràctiques pel que fa a les crides a la *API* i el tractament de dades a l'aplicació i això al final es nota quan tot va creixent i veus que va escalant i funcionant correctament. El fet de "perdre" més temps del compte en desenvolupar una funció en concret o un component que ara s'estigui utilitzant per al tractament de les dades a totes les seccions és un temps invertit per la futura eficiència de l'aplicació web i això ho he valorat molt, sobretot a la secció del *Dashboard* comparant-ho amb aquesta, que és on he vist que gràcies a l'estructuració de fitxers i funcions ara tot funciona genial.

5.6. Sprint 6

Període: 1 – 15 de Maig de 2019

5.6.1. Resum de l'Sprint

Aquest ha estat l'últim *Sprint* planificat a nivell de desenvolupament de tot el projecte (sense tenir en compte els possible imprevistos que podien sortir al final). Ha estat la secció més difícil de crear després de la dels Posts, ja que la comunicació a temps real i la gran varietat d'escenaris han fet que hagués de refactoritzar el codi fins a 3 cops.

Al final, com detallo als diferents punts de cada una de les parts desenvolupades al llarg de l'*Sprint*, he aconseguit crear una primera versió força robusta que permet assolir els punts que el client havia demanar i a l'hora també obre les portes a millores que no implicarien gaire feina. El fet d'haver crear un sistema de missatgeria amb la base ben definida permet un desenvolupament futur molt fàcil i escalable.

5.6.2. Descripció i resultat de les històries d'usuari

5.6.2.1. Base de dades

El primer que he fet ha estat dissenyar els dos models que tindran el gran protagonisme d'aquesta secció: les converses i els missatges. Per explicar-ho breument abans d'entrar en detall a la secció d'Arquitectura, cada usuari té una conversa pròpia a la Base de dades, on ell és el *sender* i l'altre usuari és el *receiver*. Cada conversa té un identificador i els missatges es relacionen a través d'aquest amb cada una d'elles.

Així doncs, els models serien els següents:

- **DB – Afegir model *Conversation***
  - Primer de tot he creat el model de la conversació amb el conjunt de camps següents:
    - *sender:* ID de l'usuari que envia els missatges.
    - *receiver:* ID de l'usuari que rep els missatges.
    - *convId: string* únic que relaciona les converses amb els missatges.

    - *status: string* indicant si la conversa ha estat llegida per l'usuari *receiver.*
- Aquest model es generarà automàticament que s'enviï el primer missatge.

❖ **DB – Afegir model *Message***

- A part, també he creat el model dels missatges. Aquests es guarden a una taula apart per tal de que les dues converses hi puguin accedir a través del *convId.* Té els següents camps:
    - *convId:* string únic que fa referència a les conversacions a les quals pertany el missatge.
    - *text: string* amb el contingut del missatge enviat.
    - *sender:* ID de l'usuari que ha enviat el missatge.
    - *status: string* que indica si el missatge s'ha rebut bé a la *API* i s'ha pogut enviar al destinatari.
    - *timestamp: string* amb l'hora a la qual s'ha enviat el missatge.
- A part, com als models anteriors de Posts i Events, també s'hi ha afegit el *flag timestamp* per tal de que la Base de dades generi i actualitzi sola els camps *createdAt* i *updatedAt.*
- Un cop creats els dos models, he començat a crear la lògica de la *API* per fer les primeres proves amb dades estàtiques.

5.6.2.2. API

❖ **API – Afegir ruta i controlador per obtenir converses**

- El primer *endpoint* que he creat ha estat el més comú i més utilitzat per totes les altres seccions (òbviament, adaptat), i és el de l'obtenció de les converses, en aquest cas.
- Cada cop que l'usuari accedeix a la secció *Dashboard (*apart de la de *Messages*, així només entrar a l'aplicació ja sabrem si tenim alguna conversa pendent de llegir) haurem d'obtenir les converses on l'usuari aparegui com a *sender*.
- La ruta necessita d'un *JWT* per validar l'usuari i així també saber quines converses cercar, gràcies a l'*ID* implícit que porta aquest *token*.

❖ **API – Afegir ruta i controlador per missatges d'una conversa**

- Un cop rebem totes les converses, l'usuari pot anar canviant entres les aquestes per enviar missatges a diferents treballadors. Cada cop que fem *click* a una conversa per mostrar els missatges anteriors i poder-ne afegir de nous, haurem de recuperar aquesta informació de la *API*.
- Per cada petició, enviarem la *convId* de la qual volem obtenir els missatges, el *JWT* de l'usuari autenticat i els retornarem ordenats de més a menys recent, d'aquesta manera els podrem tenir ordenats a la vista de l'aplicació web.

❖ **API – Afegir ruta i controlador per enviar missatges**

- Aquesta part és la que ha fet que consideri l'*Sprint* com el més difícil a nivell d'implementació de tot el projecte. El simple fet de crear un missatge i haver de pensar com ha de reaccionar tant la part del client com la del receptor, m'ha fet canviar la part de la *API* fins a 3 cops.
- Les 3 possibilitats eren viables al principi, les vaig definir per tal d'implementar-les durant l'Sprint i veure quina era la que millor s'adaptava al model actual i futur de l'aplicació. Per sort, he anat molt bé de temps i no m'ha perjudicat a les tasques següents.
- Així doncs, aquesta funció el que fa és, un cop rep un nou missatge a enviar, comprova si ja existeixen les conversacions a la *DB* (en cas que no sigui el primer missatges que s'envien aquestes dues persones, o sinó les genera en aquest moment. Crea un identificador per a les dues, l'anomenat *convId* i també guarda el missatge a la base de dades.
- Un cop hem crear/actualitzat totes les noves dades a la *DB*, hem de comprovar si cal enviar el missatge o no al destinatari. Com que la *API* coneix quins usuaris estan connectats en tot moment, si el *receiver* ho està, enviarem el missatge a través dels WebSockets, sinó simplement quedarà a la *DB*. Respecte a l'usuari que envia el missatge, enviarem un *ACK* per tal de que confirmi a la vista de l'aplicació conforme el missatge s'ha guardat correctament.
- Igual que sempre, requerirem del *JWT* per validar l'usuari, apart d'un cos del missatge vàlid (no buit) per tal de tirar endavant l'acció.

❖ **API – Afegir ruta i controlador per actualitzar l'estat de les converses**

- Per acabar la part de la *API*, he afegit la ruta per confirmar que un usuari ha llegit els últims missatges d'una conversa.
- Quan enviem un missatge i l'usuari *receiver* no està connectat, la conversa queda amb el camp *status: unread*, fent que quan l'altre usuari inicia sessió, veu que té converses pendents de llegir (surt una notificació a la barra lateral, al costat del nom *Messages).* Quan accedim a la secció i fem *click* a la conversa pendent, enviarem una petició a la *API* per actualitzar l'estat de la conversa i passar-ho a *status: read*.
- La ruta requereix de l'autenticació per *JWT*, el qual utilitzarem també per obtenir l'*ID* d'aquest usuari i així cercar la conversa més fàcilment.

5.6.2.3. Aplicació Web

❖ **WEB – Crear vista *Messages***

- Finalment queda la part gràfica de l'aplicació. La secció de vistes té una aparença molt simple i organitzada per tal de fer el més intuïtiu possible la missatgeria entre els treballadors de l'empresa.
- A la part esquerra tenim les converses actives ja començades amb altra gent, a la sobre hi trobem un cercador d'usuari per buscar entre les converses i just al costat, un botó per iniciar-ne una de nova (seleccionant a un desplegable la persona amb la qual volem començar a parlar).
- A la part central, un cop seleccionem una conversa, apareixen els missatges de cada una de les persones, separats per colors i cada missatge a un costat diferent.
- Finalment, les notificacions apareixen en forma d'un punt del color taronja (color principal de l'aplicació) al costat del nom de l'usuari a la conversa en qüestió. A part, si no som dins la secció, com he comentat abans també apareixerà el mateix punt a la barra lateral, al costa del nom de la secció *Messages.*

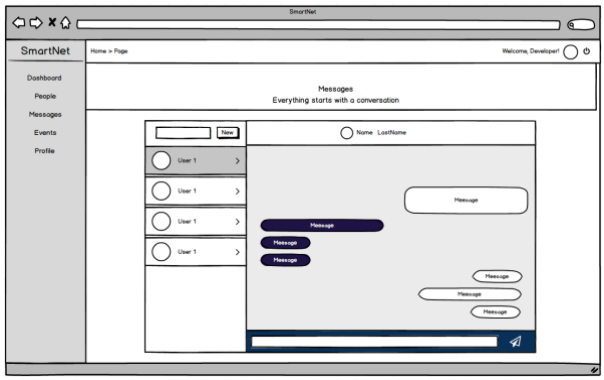


Figura 5.29: Primer disseny de la vista Messages de la plataforma.

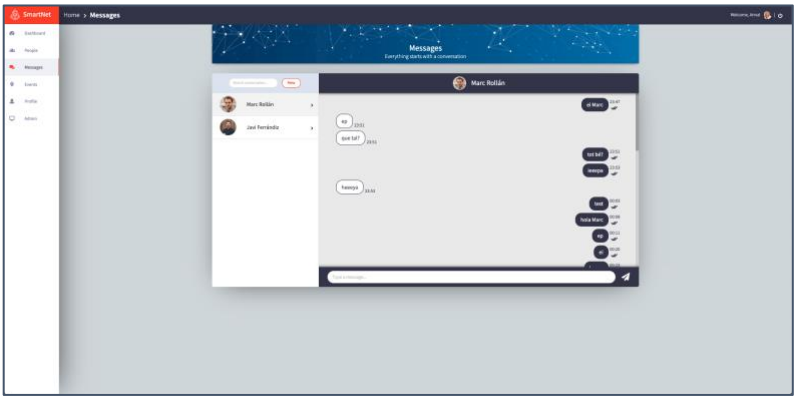


Figura 5.30: Vista finalitzada de la secció Messages.

❖ **WEB – Afegir connexió entre *Profile* i *Messages***

- A banda de crear la vista principal de missatgeria, també he adaptat algunes de les funcionalitats de l'aplicació que ja havia creat en el seu moment però faltava enllaçar un cop estigué enllestida la part final de *Messages.*
- Per una part, a la secció de *People* tenim totes les targetes que mostren un resum de cada usuari. A la part inferior hi ha dos botons, un que envia al perfil de la persona i l'altre que serveix per enviar-li un missatge. Ara que tenim la secció llesta de *Messages*, he lligat aquest botó per tal de ens porti a la secció i obri la conversa directament amb aquell usuari (tant si ja la tenim creada com si és el primer missatge que s'enviarien).
- També he adaptat el component *Main*, que era el que s'encarregava de gestionar totes les dades de l'aplicació un cop ens autenticàvem. Per tal de poder rebre missatges i les notificacions mentre estem navegant per l'aplicació (sense ser necessàriament dins la secció *Messages*), és necessari que les converses es carreguin al primer moment. D'aquesta manera les guardarem i

les podrem anar actualitzant, veient si hi ha canvis cada cop que entra un nou missatge per WebSockets.

- **WEB – Afegir notificacions a la barra lateral de navegació**
  - Seguint amb el punt anterior i com ja he explicat abans, he introduït les primeres notificacions a temps real a la barra lateral de navegació. Així doncs, un punt del color principal de l'aplicació al costat del nom de cada secció ens informarà que tenim un avís pendent en aquell lloc. Penso que és una manera força ràpida de localitzar el motiu de l'avís i saps al moment de quina secció pertany.
  - Ara per ara només funciona amb la secció de *Messages*, però la lògica ja l'he creat pensant en la introducció d'aquesta en altres seccions sense cap problema.

- **WEB – Redissenyar mínimament les capçaleres de les seccions**
  - A mesura que anava dissenyant la part de l'aplicació web d'aquest secció de *Messages* vaig veure que les capçaleres que havia crear de bon principi no acabaven de quadrar i ocupaven molta pantalla. Aquesta secció requereix de tot l'espai possible i això em va fer pensar que potser era necessària donar una volta al disseny i a com estaven estructurades.
  - He afegit una mica més de lògica al component que crea les capçaleres per tal de fer-les més petites, mantenint la informació que proporcionen intacta i a l'hora deixant més espai lliure pel contingut de cada secció.

- **WEB – Afegir el nombre total de *Posts* creats per usuari a la secció *People***
  - A les mateixes targetes dels usuaris on he afegit el *link* cap a la secció de *Messages*, també hi havia una dada que no estava actualitzada: el nombre total de *Posts* que un usuari ha publicat.
  - Això m'ha fet crear un camp nou al model de la base de dades, a la col·lecció User:

    - *numPosts*: enter que conté el nombre total de *Posts* creats per l'usuari.
- Ara, cada cop que l'usuari afegeixi o elimini un post, aquesta variable s'actualitzarà i podrem mostrar aquest valor a la seva targeta resum.

5.6.3. Conclusions de l'Sprint

Per fi he deixat llesta l'aplicació web pel que fa als requeriments bàsics que em vaig proposar des d'un inici. Com ja vaig comentar, sabia perfectament que acabaria els *Sprints* abans d'hora i això em permetria tenir un marge per arreglar coses o millorar-les, i així ha estat.

Aquest *Sprint* al principi era el penúltim, *Events* havia d'haver estat l'última secció que dissenyés, però durant els primers *Sprints* ja vaig anar veient que les coses s'acabarien complicant a la secció de *Messages*. I així ha estat.

També m'ha servit molt per acabar de tancar petits detalls d'altres seccions que venien relacionades amb aquesta i que a l'hora m'han permès ajustar i millorar d'altres que no tenia previstes. Ara podria dir que ja he acabat totalment l'aplicació, però també es veritat que amb el temps que tinc de marge intentaré donar un repassada general per veure si puc aportar alguna millora o funcionalitat que no havia estat plantejada i així rebre un *feedback* del client per a una possible futura millora que pogués incorporar.

6. Implementació

6.1. Tecnologies

Per tal de dur a terme tot aquest desenvolupament per a la creació del projecte, vaig haver d'escollir un seguit de tecnologies que em permetien des de gestionar totes les dades fins a mostrar-les a l'aplicació web. Hi ha tendències que fa molts anys que duren i altres que evolucionen i van competint per ser la més utilitzada dins del seu àmbit, i això fa que a l'hora d'escollir quines d'aquestes fer servir sigui important estudiar-les mínimament per sobre.

Dit això, he escollit les tecnologies més importants en cada sector del projecte i he comparat els seus avantatges respecte els inconvenients, per tal de fer-me una idea de què podria anar millor a l'hora de crear l'aplicació i així també poder escollir un conjunt d'eines que tinguessin futur dins d'aquest món. Així doncs, he dividit la comparació en els mateixos tres blocs que porto explicant al llarg del treball:

- DB: SQL vs NoSQL
- API: Laravel vs ExpressJS
- WEB: React vs Angular vs Vue

6.1.1. DB

Per a aquesta secció, escolliré un exemple de cada estructura de dades (*MySQL* per *SQL* i *MongoDB* per *NoSQL*) per tal d'avaluar quina de les dues és millor per aquest treballar i per què he escollit MongoDB.

És una base de dades molt consolidada, amb una gran comunitat, proves extensives i estabilitat.	Proporciona flexibilitat per canviar l'esquema de dades sense modificar cap de les dades existents.
Està disponible per a totes les plataformes principals, incloent-hi Linux, Windows, Mac, BSD i Solaris.	És escalable horitzontalment, ajudant a reduir la càrrega de treball i donant la possibilitat d'escalar amb facilitat.
És *open-source* i gratuïta.	La base de dades no requereix un administrador de base de dades. Com que és fàcil d'utilitzar d'aquesta manera, els desenvolupadors i els administradors poden utilitzar-lo.
Es pot replicar a través de múltiples nodes, reduint la càrrega de treball i augmentant l'amplitud i disponibilitat de l'aplicació.	Té una molt bona *performance* amb tota classe de peticions senzilles.

Donades totes les possibilitats i avantatges de les dues bases de dades, és molt clar que l'aplicació es podria haver creat amb qualsevol d'elles, per alguna cosa són les principals referències quan parlem de bases de dades. Però hi ha un punt en concret que va ser suficientment important per decidir utilitzar *MongoDB* respecte a *MySQL*:

"Proporciona flexibilitat per canviar l'esquema de dades sense modificar cap de les dades existents."

Les dues bases de dades funcionen bé pel que fa a l'escalat, la gestió de dades i la comunitat que porten a darrere, però quan parlem d'aplicacions a desenvolupar que podem arribar a escalar molt i variar durant aquest procés, hem de fixar-nos bé en què

passarà amb totes les dades existents que tenim fins llavors i com les seguirem tractant en un futur.

Aquest cas és molt típic en aquestes situacions, en el qual durant el procés de disseny de noves funcionalitats cal implementar canvis (afegint nous camps per calcular noves dades) que poden xocar amb tot el que es portava dissenyat fins aleshores. *MongoDB* ofereix la possibilitat de poder tenir documents amb camps totalment diferents dins d'una mateixa col·lecció, fent que la intel·ligència passi a una capa superior (*API*). D'aquesta manera, com seria un cas d'exemple aplicat a aquest projecte, el fet d'introduir un nou camp dins d'una col·lecció plena de dades anteriors fa que simplement hàgim de tenir en compte a la *API* que certs valors d'aquesta col·lecció poden tenir un valor nou o diferent, fent un desenvolupament molt més flexible i còmode.

A part, el fet que degut a la seva facilitat d'ús no es requereixi un administrador de base de dades a l'equip, també facilita el fet de poder assignar tasques a una persona que pugui a l'hora tocar la *API*, per exemple, ja que no deixa de ser molt semblant a nivell de documents i estil JSON.

Per tant, donat l'anàlisi anterior, he decidit utilitzar **MongoDB**.

6.1.2. API – Frameworks de Back-End

Per la part de la API també compararé dos dels *frameworks* més utilitzats: *Laravel* i *ExpressJS*. Els dos es poden connectar perfectament amb MongoDB com a base de dades principals, així que no hi ha cap problema en aquest sentit i faré l'anàlisi bastant-me en altres principis:

	
Permet una creació i desenvolupament ràpid d'aplicacions.	Permet una creació i desenvolupament ràpid d'aplicacions.
Té una molt bona *performance* a l'hora d'executar aplicacions web.	Molt bona gestió respecte a operacions I/O.
És fàcil d'aprendre i utilitzar.	Escrit en JavaScript, la qual cosa minimitza l'esforç en contractar equip especialitzat en un altre llenguatge.
Té una documentació fàcilment assequible.	Comunitat *open-source*.
Compleix els requisits per a les aplicacions web modernes (temes de seguretat...).	Fàcil integració de serveis i middleware de tercers.

Com podem veure, els dos *framework* són molt potents per a crear aplicacions web a la part de servidor, com seria el nostre cas de la *API*, facilitant molt la creació i desenvolupament d'aplicacions i tenint un gran suport gràcies a tota la comunitat que dia rere dia millora la base dels dos projectes. Els dos també compleixen amb una tasca molt important d'avui en dia pel que fa a les aplicacions de servidor: la seguretat. Al final, tota aplicació que tracti amb dades compromeses pels usuaris ha de gestionar-les de forma segura i tractar aquestes dades a la part del servidor, mai podem confiar que l'aplicació web que dissenyem sigui infal·lible i ens retorni les dades d'un client correctament. Per aquest motiu, ja sigui a nivell de llibreries externes incloses dins del mateix *setup* del projecte o incloent *middlewares*, tindríem aquesta part totalment solucionada, la qual cosa implica una gran part de la feina feta i que es va actualitzant dia a dia gràcies a les millores que tota la comunitat aporta.

Finalment doncs, els punts clau que em va fer decidir optar per *ExpressJS* a l'hora d'escollir un *framework* pel Back-End, van ser els següents:

- A diferència de *Laravel*, el qual es programa amb PHP, *ExpressJS* és un *framework* de *NodeJS* escrit en *JavaScript*. Ja que els principals *framework* de Front estan escrits també en *JavaScript* (tal com explicaré a continuació), és molt més fàcil per a qualsevol desenvolupador que ja hagi tocat aquest llenguatge algun cop, poder treballar-hi. A part, com explicava a la taula d'avantatges, és molt més fàcil per l'equip i rendible a llarg termini que una persona pugui tocar les dues coses, ja sigui per casos puntuals on hi hagi necessitat de desenvolupar una funcionalitat nova a l'aplicació web i gran part de l'equip s'hi hagi de dedicar, com al revés, si en algun moment falla algú i s'ha de substituir temporalment, qualsevol persona podria ocupar aquest lloc.
- I el segon punt, i no per això menys important, és la gran *performance* que té *ExpressJS* amb les operacions d'entrada i sortida de missatges. Ja que el plantejament inicial era construir una *API*, aquestes es caracteritzen per estar donant i rebent dades constantment, i òbviament necessitava trobar un *framework* que s'adaptés a aquestes característiques.

Donades les següents condicions analitzades, he optat per fer servir *ExpressJS* per al desenvolupament de la *API* que s'executarà a la part del servidor.

6.1.3. WEB – Frameworks de Front-End

Finalment queda la part de l'aplicació web. Aquesta ha estat la part més difícil a l'hora d'escollir quin *framework* o llibreria utilitzar per dissenyar l'aplicació, ja que com mostraré a la següent taula comparativa, tots són molts semblants entre si. Una de les principals característiques d'aquests 3 principals *Frameworks* (Angular, Vue i React, tot i que aquest últim és una llibreria, no un *framework*) és que els 3 ho fan tot molt bé i molt semblant, donant la possibilitat a l'usuari de poder escollir treballar amb el que se senti més còmode i millor s'adapti a la resta del seu projecte. Dit això, veiem les principals característiques:

		
Gran comunitat i manteniment a càrrec de Google.	Llibreria amb la comunitat més gran i a càrrec de Facebook.	Projecte sostingut per Laravel. Moltes grans empreses l'estan començant a adoptar.
Té una metodologia d'implementació perfectament definida i establerta.	Gran ecosistema i llibreries (Redux).	Simplicitat a la sintaxi i corba d'aprenentatge molt simple per als principiants.
És un framework que segueix el patró *MVC*.	Flexible i escalable.	Estructura del codi molt senzilla
Estructura i arquitectura creades específicament per a una gran escalabilitat al projecte.	Corba d'aprenentatge mitjana.	Ha adoptat molts dels conceptes que han triomfat a Angular i React.
	Molt bona performance gràcies al seu *Virtual-DOM*.	

Igual que passava amb els *Frameworks* de Back-End, al Front ens trobem algo molt semblant. Com deia, totes tres opcions són perfectament vàlides per a dissenyar aquesta aplicació, totes estan escrites amb JavaScript i totes s'entenen perfectament amb la *API* que he creat. Per tant, els motius pels quals he escollit React són els següents:

- Primerament, React no és un *framework*. React és una llibreria per a dissenyar interfícies d'usuari, la qual cosa implica que no porta tot el conjunt de paquets i dependències que porta Angular, per exemple. Això redueix molt la mida total del codi final pujat a producció i dóna la llibertat de poder anar afegint les llibreries segons la necessitat del moment. Per aquesta raó, el dubte quedaria entre React i Vue.

- Vue i React són molt semblants entre si. De fet Vue ha utilitzat moltes de les grans eines que utilitza React (com el *Virtual-DOM* i la llibreria *Redux*, que facilita la centralització de l'estat de l'aplicació en un sol lloc) i les ha redefinit lleugerament per introduir-les a la seva arquitectura. La gran diferencia entre ells, és que Vue, a part de ser un *framework* (no tant pesat com Angular, però sí que conté forces dependències) està en procés de creixement i ara per ara només està recolzat fermament per una persona (òbviament, amb una comunitat que cada cop creix més). React, per la seva banda, va ser crear i segueix sent millorat per Facebook i això dóna més tranquil·litat a l'hora de saber que t'estàs enfocant a fer una aplicació amb un codi que saps que, com a mínim a curt termini, tindrà un gran suport a l'esquena.

- Finalment, a la decisió final també hi vaig tenir en compte la meva experiència professional. He treballat forces mesos amb React i m'hi sento molt còmode programant. He estudiat i desenvolupat petits projectes amb els altres dos *frameworks* i he pogut veure quin dels 3 m'agradava més. A banda d'això, React és la llibreria més sol·licitada al mercat a nivell d'ofertes de desenvolupadors Front-End, i això fa que creixi tant el nombre de gent que s'interessa per la llibreria com la comunitat que es dedica a millorar perquè tot segueixi així de bé.

Així doncs, cal recalcar que l'aplicació s'hagués pogut dissenyar perfectament amb les 3 opcions comentades, però al final se n'havia d'escollir una. Per tant, donats els punts argumentats, la decisió ha estat escollir **React** com a llibreria per a implementar l'aplicació web a nivell de Front-End.

6.2. Arquitectura

Un cop escollides les tecnologies per implementar tot el projecte, cal revisar com ha estat el desenvolupament dels 3 grans blocs. Per fer-ho més simple, els he separat de la mateixa manera que ho porto fent fins ara, així podré entrar més en detall i posar exemples més concrets.

6.2.1. Base de dades

La base de dades és el punt principal d'aquest gran projecte, dins el qual tenim totes les dades que es tracten a la *API* i es visualitzen a l'aplicació web. Té un total de 5 col·leccions, les quals han acabat amb les següents estructures:

- Users

Aquesta col·lecció té les dades principals de tots els usuaris de l'aplicació i està formada pels següents camps:

Camp	Tipus	Descripció
username	*String*	Nom especial que té l'usuari dins l'aplicació.
name	*String*	Nom real de l'usuari.
lastName	*String*	Cognom real de l'usuari.
email	*String*	Correu electrònic de l'usuari.
password	*String*	Contrasenya (encriptada) de l'usuari.
numPosts	*Number*	Nombre total de publicacions que l'usuari ha creat.
status	*String*	Descripció curta personalitzada.
age	*Number*	Edat de l'usuari.
city	*String*	Ciutat on resideix l'usuari.
mobile	*String*	Numero de telèfon de l'usuari.
role	*String*	Posició que ocupa dins de l'empresa.
level	*Number*	Nivell de l'usuari que decideix quines accions pot fer.
aboutMe	*String*	Descripció general personalitzada sobre l'usuari.
profileImg	*String*	Direcció URL on està guardada la foto de perfil de l'usuari.
token	*String*	Clau secreta per registrar l'usuari a l'aplicació.
weather	*Object*	Objecte que conté totes les dades del temps de la ubicació on s'ha connectat l'usuari a l'última hora.
joinedOn	*Date*	Data a la qual l'usuari es va unir a l'aplicació.
updatedOn	*Date*	Data de l'última actualització de dades de l'usuari.

- Posts

Totes les publicacions que fan els usuaris es guarden a dins d'aquesta taula. Tenim els següents camps:

Camp	Tipus	Descripció
creator	*Object ID*	Referència de l'ID de l'usuari que ha creat la publicació
text	*String*	Contingut de la publicació
likes	*Array*	Array amb el conjunt d'ID's dels usuaris als quals els hi ha agradat la publicació.
comments	*Array*	Array amb el conjunt de comentaris associats a la publicació. Cada comentari té un text, la data de quan va ser creat i l'ID de l'usuari que el va escriure.

A aquesta taula ja comencen a aparèixer els tipus *Object ID*, que són IDs que fan referència a un document en concret d'una altra col·lecció (en aquest cas la de *Users)*. Això serveix per poder obtenir les dades de l'usuari que ha creat una publicació o comentari i tenir-les actualitzades. De l'altra manera, si guardéssim les dades de l'usuari al moment que fa la publicació, per exemple, si aquest usuari canviés de nom o de foto de perfil, no les podríem tenir actualitzades (o probablement hauríem de tenir un procés que actualitzes **totes** les col·leccions amb les noves dades, cosa que no és gens eficient).

- Events

A aquesta col·lecció guardarem cada un dels esdeveniments que són creats pels usuaris de l'aplicació. Igual que a les col·leccions anteriors, tenim forces referències dels usuaris, de tal manera que podrem obtenir les seves dades actualitzades cada cop que consultem cada un dels esdeveniments. La col·lecció està formada pels següents camps:

Camp	Tipus	Descripció
name	*String*	Nom de l'esdeveniment
description	*String*	Descripció de l'esdeveniment

location	*String*	Latitud i longitud del lloc on se celebrarà l'esdeveniment.
host	*Object ID*	Referència de l'ID de l'usuari que ha creat l'esdeveniment.
type	*String*	Tipus d'esdeveniment (Sopar, dinar...)
dateFrom	*Date*	Data a la qual comença l'esdeveniment.
dateTo	*Date*	Data a la qual acaba l'esdeveniment.
guests	*Array*	Array amb el conjunt d'ID's dels usuaris que assistiran a l'esdeveniment.

- Conversations

La part de *Messages* està dividia en dues parts. Per un costat tenim la col·lecció de les *Conversations*, la qual guarda els següents valors:

Camp	Tipus	Descripció
sender	*String*	Referència a l'ID de l'usuari que consta com a remitent.
receiver	*String*	Referència a l'ID de l'usuari que consta com a destinatari.
convId	*String*	Identificador únic de la conversa, el qual serveix també per identificar els missatges.
status	*Object ID*	Estat de la conversa respecte al remitent. Pot constar com a *read* o com a *unread*, depenent si l'usuari destinatari ha llegit els últims missatges enviats.

Aquesta col·lecció guarda totes les converses que es generen a la secció de *Messages* quan un usuari inicia una conversació amb un altre. Com es pot intuir, sempre tindrem dues converses amb el *sender* i el *receiver* invertits, ja que cada usuari ha de tenir una conversa on ell sigui el remitent, per saber l'estat dels missatges respecte a l'altra persona i, a part, també ens servirà per si mai un usuari elimina la conversa, que només elimini el seu document de la col·lecció, de tal manera que l'altre usuari pugui seguir consultant la conversa.

- Messages

La segona col·lecció respecte a la part de *Messages* de l'aplicació. Aquí és on es guarden cada un dels missatges que s'intercanvien entre els usuaris. Tenim els següents camps:

Camp	Tipus	Descripció
convId	*String*	Identificador únic que fa referència a la conversa a la qual pertany aquest missatge.
text	*String*	Contingut del missatge.
sender	*Object ID*	Referència a l'ID de l'usuari que ha enviat el missatge.
status	*String*	Estat del missatge per saber si s'ha enviat correctament i la *API* l'ha rebut i guardat a la base de dades. Podem tenir *sent* o *success.*
timestamp	*String*	Data (en aquest cas guardada número total de mil·lisegons) a la qual va ser enviat el missatge.

Cada un dels missatges té un identificador de conversa (el qual es comparteix amb dos documents, tal com he explicat a la col·lecció anterior de *Conversations*). D'aquesta manera, quan obri la secció de *Messages* des de qualsevol usuari, recuperarem tots els missatges, tant els que ha enviat ell com els que ha rebut. Aquest és l'únic cas del projecte on no guardem els identificadors dels missatges dins d'un *Array*, sinó que he creat una col·lecció dedicada per tal de poder accedir-hi des d'altres llocs. La raó, a part del que he comentat ara sobre les converses i que cada usuari hi accedeix a través del seu *convId*) és que els missatges no poden variar, un cop un missatge és creat, no hi ha manera d'editar-lo. Per tant, podem guardar-lo a una col·lecció diferent sabent que no caldrà estar agafant-los sempre que consultem les converses, ja que el seu contingut no variarà i no haurem de patir per tenir la seva última versió.

6.2.2. API

La part de la *API* és on concentra tot la lògica que treballa amb les dades que hem vist anteriorment. Tal com està dissenyada, tenim 3 mòduls principals: el model, les rutes i els controladors.

6.2.2.1. Models

Els models són els fitxers que connecten la base de dades amb la *API*, els quals defineixen l'estructura de cada col·lecció i permeten interactuar amb elles. Per tant, n'hi ha d'haver un per cada col·lecció existent a la base de dades.

Per posar un exemple i veure com està format, agafarem el model de la col·lecció *Events*:

```
const event_schema = new Schema({
    name: String,
    description: String,
    location: String,
    host: { type: Schema.Types.ObjectId, ref: 'User' },
    type: String,
    dateFrom: Date,
    dateTo: Date,
    guests: [{ type: Schema.Types.ObjectId, ref: 'User' }], // Array of ObjectIDs from users who are assisting
the event
}, { timestamps: true });

module.exports = mongoose.model('Event', event_schema);
```

Definim l'esquema de la col·lecció de la base de dades amb tots els seus camps i el tipus que els hi correspon i al final de tot exportem l'objecte creat amb el nom amb el qual volem anomenar la nova col·lecció. D'aquesta manera, a l'exportar-ho fem que tota la *API* pugui tenir accés a la col·lecció de la base de dades i pugui fer-hi les consultes necessàries.

Cal afegir que, si ens hi fixem, l'exportació es fa amb el nom "*Event*", però MongoDB sempre agafa els noms i els passa a minúscula i plural, així evita confusions i segueix una

estructura uniforme (si mirem a la base de dades, hi veurem la nova col·lecció com a *events*).

6.2.2.2. Rutes

La següent part del procés passa per les rutes. Quan fem una petició a la *API*, la fem a través d'una URL. És aquí on les rutes s'encarreguen de dirigir el tràfic i saber quin controlador serà necessari per executar les tasques necessàries que té associat aquest URL donat.

Seguint amb l'exemple dels *Events*, veurem com està formada una ruta d'exemple. En aquest cas, la URL que com a usuari demanaríem des de la lògica de l'aplicació web o qualsevol altre programa seria la de voler obtenir **la llista de tots els *Events* creats.**

L'URL d'exemple seria la següent:

*host:port**/event/get_events***

Primer de tot definim a quin fitxer de rutes ens hem de dirigir per tal de gestionar la petició:

```
app.use('/event', event_routes);
```

Amb aquesta línia diem que tot URL que comenci amb "*/event*" es tracti al fitxer declarat com *event_routes*, el qual té **totes** les rutes possibles que podem gestionar referents a la secció d'esdeveniments.

Un cop al fitxer, hauríem de seguir buscant la ruta concreta que coincidís, en aquest cas seria la següent:

```
router.get('/get_events', isAuth, eventsController.getEvents);
```

Per acabar, trobaríem la ruta total que acabaria completant la URL, en aquest cas, la petició hauria de ser GET (per rebre dades de tornada). Un cop coincideix el resultat, el funcionament va d'esquerra a dreta: primer coincidim amb la part de la URL, després

executem tots els *middlewares* fins que arribem a l'últim paràmetre de la funció, que és el que crida a la funció del controlador que s'encarregarà de tota la part del tractament de les dades que ha de retornar. Els *middlewares* són tot tipus de funcions que s'executen abans d'arribar el controlador. En aquest cas en tenim un que es diu *isAuth*, que el que fa és comprovar si rebem un *JWT* a les capçaleres de la petició per autenticar l'usuari i donar-li permís per rebre les dades.

Aquesta és una estructura clàssica i és la que més he utilitzat al llarg de tot el desenvolupament de la *API*.

6.2.2.3. Controladors

Finalment, l'última part del camí abans de retornar les dades cap al client és la dels controladors. Seguint l'exemple anterior, el controlador que cridem serà el d'*Events* i la funció en concret serà la de *getEvents()*:

```
exports.getEvents = (req, res, next) => {
  Event.find()
    .sort('dateFrom') // First sort
    .populate('host', '_id name lastName profileImg') // Populate the host user
    .populate('guests', '_id name lastName profileImg') // Populate the guests array with all its user info
    .then(events => {
      res.status(200).json({ events, message: 'Events fetched.' });
      console.log(`[API] - (GET) /event/get_events - Status 200 - ${events.length} Events fetched!`);
    })
    .catch(err => {
      if (!err.statusCode) {
        err.statusCode = 500;
      }
      next(err);
    });
};
```

El que fa aquesta funció és buscar tots els resultats de la col·lecció *Events* amb el mètode *find()* sense cap paràmetre, ja que no els volem filtrar. A continuació, ordena els resultats pel camp *dateFrom* (de menys a més recent), després fa un *populate()* dels camps *host* i *guests*, la qual cosa significa que omplirem els resultats amb les dades completes de tots els usuaris que tenien el seu ID referenciat a aquests camps, i finalment retornarem els *events* trobats cap al client amb un *status 200* si tot ha anat

bé. En cas contrari, a totes les funcions de tots els controladors hi ha un mètode *catch()* que s'encarrega de gestionar els errors i llençar-los correctament.

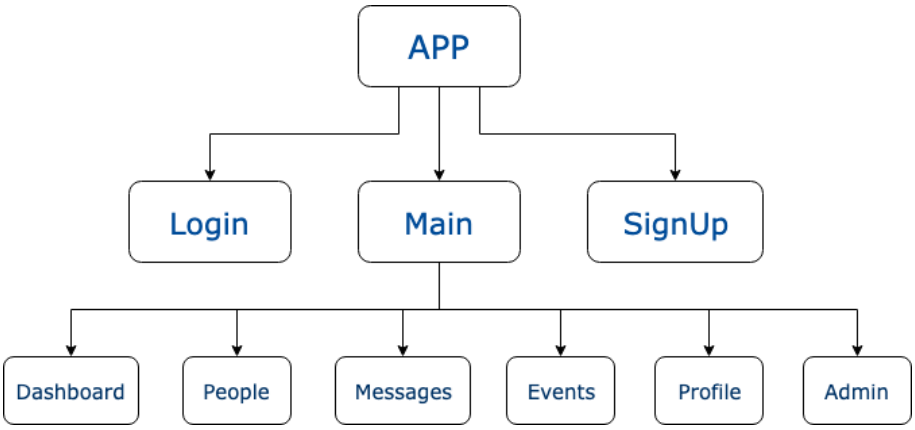
Un cop arribat aquest punt, la *API* dóna per finalitzada la gestió de la petició i segueix a l'espera de noves peticions. Ara doncs, la feina la té el client (en el nostre cas l'aplicació web), que és qui ha demanat les dades i les ha de muntar per mostrar-les a la web.

6.2.3. Aplicació Web

Per últim, l'aplicació web és qui s'encarrega de mostrar i gestionar les dades demanades a la *API* a la part visual del navegador del client. En aquest cas, com ja vaig explicar a l'inici, he dissenyat una *SPA*, cosa que implica que la part de l'aplicació ha de tenir una mica més d'intel·ligència que una aplicació web tradicional, ja que ha de saber gestionar cada un dels apartats o seccions per si mateixa i saber quines dades i en quin moment les ha de mostrar o actualitzar.

La principal característica de les *SPA (Single Page Applications)* és que només demanen un arxiu *.html* al servidor el primer cop que ens hi connectem a través de l'URL mitjançant qualsevol navegador. De tal manera, que si anem canviant de seccions, és l'aplicació la que s'encarrega de canviar tota la interfície i així "simular" a l'usuari que està navegant, com si estiguéssim comunicant-nos amb el servidor i intercanviant més arxius *.html*. Per aquesta raó, és molt important escollir una estructura adequada per poder gestionar tot el flux de dades i cada una de les seccions que voldrem crear de manera eficient i entenedora. Així doncs, he creat un conjunt de components que s'encarregaran de gestionar tasques concretes i guardar les seves respectives dades. L'esquema final és el següent:

6.2.3.1. Components principals i gestió de l'estat



Com explicava, he dividit l'aplicació amb un conjunt de components que per ells mateixos gestionen les dades i la lògica d'aquestes per tal de mostrar a l'usuari la secció corresponent que toqui a cada moment. Tenim un total de 10 components, els quals es divideixen en els següents:

- **APP**: Component principal de l'aplicació. És el component pel qual sempre passarem abans de decidir què mostrar a l'usuari. Gestiona l'estat d'autenticació de l'usuari, sap si s'ha autenticat, guarda les seves dades per tal que qualsevol altre component de l'arbre les pugui utilitzar i és qui decideix si cal mostrar una secció de *Login* o *SignUp* o simplement delegar al component inferior *Main* la pròxima tasca d'escollir què vol veure l'usuari.

- **Login**: Quan el component *APP* detecta que no hi ha cap sessió activa al navegador o l'última que hi havia ja ha caducat (cada sessió dura un total de 4h), delega la gestió a aquest component, que s'encarrega de renderitzar una vista a través de la qual l'usuari es pot autenticar. Un cop l'usuari introdueix les dades correctament, aquest component comunica tota la informació de tornada cap al component *APP*, on queda guardada per les pròximes 4 hores.

- **SignUp**: De la mateixa manera que el component *Login* anterior, quan el component *APP* no detecta cap sessió vàlida, delegarà la gestió al component *SignUp* si l'usuari no té cap compte vàlid encara i es vol registrar. Un cop

introduïdes les dades, funciona exactament igual que abans: retornem les dades cap al component *APP*, les guardem i seguim amb el pròxim component *Main*.

- **Main**: Quan tenim una sessió activa, aquest component és qui s'encarrega de determinar quina secció està demanant l'usuari segons l'URL introduït i així delegar al component corresponent tota la feina de renderitzar la vista i les dades. Aquest component és molt important, ja que com he explicat durant els *Sprints*, és qui s'encarrega de guardar les dades que l'usuari va demanant a la *API* des de qualsevol secció. Com he dit, és el "pare" que sap què demanar l'usuari i per tant "veu" a totes les seccions a un nivell inferior per sota seu, d'aquesta manera pot emmagatzemar les dades per si l'usuari va canviant entre seccions i requereix unes dades que havia obtingut fa una estona a un apartat diferent. Això fa molt eficient tota la gestió de crides que es realitzen a la *API*, ja que podent guardar les dades dins aquest component minimitza **molt** el nombre total de crides i allibera les connexions per tal de donar prioritat a altres usuaris.

- **Dashboard**: Aquest és el primer component que gestiona tota la lògica d'una secció. És la primera secció que carrega un cop ens autentiquem i mostra el resum de totes les novetats de l'aplicació, des de les últimes publicacions fins als pròxims esdeveniments. Quan es crea el component, es crida a la *API* per obtenir tant les publicacions com els esdeveniments, els quals són enviats cap al component superior *Main* per a ser guardats un cop s'han mostrat a l'usuari.
  Aquest és l'exemple que posava abans. Si ara volguéssim dirigir-nos a la secció d'*Events* no caldria tornar a demanar-los tots a la *API* (a no ser que fos la primera pàgina que visitéssim). Al passar pel *Dashboard* i obtenir-los, quan el component *Main* detecti que volem visitar la secció d'*Events*, comprovarà si tenim els esdeveniments emmagatzemats i els passarà cap al component *Events* corresponent per tal que els mostri. Aquí és on hi ha la gràcia del component *Main.*

- **People**: Segon component de seccions, el qual mostra totes les persones que treballen a l'empresa. Apareixen en forma de targeta amb un petit recull de la

informació més important de cada usuari juntament amb dos botons, un per visitar el seu perfil i l'altre per enviar-li un missatge directe.

Aquesta és una secció diferent pel que fa a les dades guardades i les crides a la *API*. Aquí vaig decidir que cada cop que es consultés la secció, es realitzes una crida a la *API* per obtenir les últimes dades dels usuaris registrats. També podria haver seguit el guió igual que es fa amb les publicacions del *Dashboard*, però com que no era un requisit mínim per part del client, vaig seguir aquesta aproximació. Per tant, el llistat de persones no es guardarà mai al component *Main*.

- **Messages**: Tercer i un dels més complicats de crear. Aquest component s'encarrega de gestionar la secció de missatgeria que, com a requisit del client, havia de ser a temps real. Per aconseguir aquest propòsit, el component treballa juntament amb el *Main* per gestionar tot el flux de dades i així poder aportar aquesta entrada de dades automàtica cada cop que rebem un missatge.

  Així doncs, el component *Main* sap en tot moment quina és la conversa que té activa l'usuari i alhora està escoltant a la *API* (que és qui ens enviarà els missatges) per saber si hem de passar-ne un de nou cap al component *Messages* i que aquest el mostri per pantalla.

  Seguint amb el mateix concepte de sempre, aquí el component *Main* és qui s'encarrega de gestionar l'entrada i sortida de missatges, mentre que el component *Messages* és qui controla el tema de les converses actives, missatges a mostrar i creació de nous xats. A part, com que el component *Main* està per sobre de totes les seccions, això permet que pugui estar escoltant nous missatges mentre nosaltres com a usuaris estem navegant pel nostre perfil o pel *Dashboard*, donant-nos la possibilitat d'avisar a través d'una notificació quan rebem un missatge nou "en segon pla".

- **Events**: Quart component principal obert a tots els usuaris, on es mostren els esdeveniments que són creats pels propis treballadors de l'empresa. Hi ha un seguit de targetes que mostres un resum dels detalls més importants de l'esdeveniment, juntament amb el dia i l'hora, el lloc i una capçalera amb una foto que defineix el tipus d'ocasió que s'ha planificat. És molt semblant al

component *Dashboard*, ja que carrega en aquest cas el conjunt d'*Events* i els guarda al component *Main* per a futures visites i per a la utilització d'altres components que els necessitin, com per exemple el *widget* d'*Events* que hi ha al *Dashboard*, el qual mostra els 5 esdeveniments més propers donada la data d'avui**.**

- **Profile**: El cinquè i últim component públic de l'aplicació. Aquí és on es pot veure tota la informació tant del nostre propi usuari com de qualsevol altre de l'empresa. Recull tota la informació de l'usuari, les seves publicacions i els pròxims esdeveniments que aquest ha creat.

  Aquest és un altre exemple de component que fa ús tant de les publicacions com dels esdeveniments que s'han demanat a la *API* i guardat al component *Main* a un primer moment. D'aquesta manera els tindrem llestos quan accedim a un perfil sense haver de tornar a fer crides externes a la *API*.

  L'única crida que cal fer, potser, és la d'obtenir informació bàsica de l'usuari al qual li volem visitar el perfil. Quan ens autentiquem a la plataforma, la nostra informació queda guardada al component *APP*, tal com he dit abans. Per això, cada cop que visitem el component (no secció!) *Profile*, hem de diferenciar si ho estem fent per veure la nostra pròpia informació (en aquest cas no cal consultar la *API*, simplement demanant la informació al component *APP* ja podríem mostrar la pàgina) o bé estem accedint al perfil d'un treballador de l'empresa. En aquest últim cas sí que hauríem d'accedir a la *API* per obtenir la informació cada cop que canviéssim de treballador, ja que aquestes dades no es guarden al component *APP* ni *Main.*

- **Admin**: Per últim tenim el component *Admin*, el qual només està disponible pels usuaris que tinguin nivell 1 o superior. Aquest component és l'encarregat de gestionar el registre de nous usuaris a l'aplicació, creant nous tokens i visualitzant quins usuaris queden pendents a registrar-se.

  És un component força simple pel que fa al tractament de les dades que gestiona, però a l'hora també va ser el primer pas per desenvolupar components que només fossin accessibles amb un cert nivell d'usuari. Com he dit a l'inici d'aquest

punt, a l'haver de gestionar tot des de l'aplicació (ja que estem creant una SPA) també s'ha de tenir molt clar i definit quins apartats són accessibles per cada grup de treballadors de l'empresa. La *API*, que és qui al final dona l'últim vist i valora si l'acció que es vol realitzar és vàlida o no, és qui té l'última para i ho valida tot a nivell de Back-End, però per la part de l'aplicació també volia proporcionar un nivell de seguretat òptim i viable.

7. Conclusions i treball futur

Finalment podem dir que l'aplicació ha tancat la seva fase inicial, resolent al complet tots els requisits establerts pel client i creant una primera versió del projecte molt satisfactòria. Per tal de resumir i veure els detalls de com he aconseguit resoldre cada un dels punts, seguint amb la mateixa estructura de tot el treball, aquestes són les solucions / implementacions que s'han dut a terme a cada una de les 3 àrees:

- Pel que fa a la Base de Dades, no s'ha generat un gran nombre de col·leccions ni dades, aprofitant al màxim els documents en format *JSON* dins les col·leccions per tal de tenir ràpid accés a les dades que no s'hauran d'editar en un futur i per tant poden conviure a dins d'un document (no col·lecció), com seria un missatge dins d'una conversa o un comentari dins d'una publicació. Per altra banda, també he utilitzat les referències a diferents taules per tal de no repetir dades innecessàriament, alleugerint el nombre d'informació a guardar duplicada i càrrega del servidor.
- A la *API* s'han generat tot un seguit de rutes molt específiques per a realitzar feines molt puntuals. Per què serveix això? Per tal de simplificar al màxim la lògica de les funcions i així oferir forces més alternatives per a desenvolupar qualsevol cosa sense estar lligat a una ruta concreta que retorna unes dades en concret. D'aquesta manera, si es creen funcions molt més genèriques, permets centrar tot el procés creatiu a la part de l'aplicació web o qualsevol altre tipus de plataforma per a la qual s'estigui desenvolupant l'aplicació, obrint les portes a futures millores i nous productes (aplicacions per telèfons mòbils, etc).
- Finalment quedaria la part de l'aplicació, que és on es van demanar més requisits i on més temps hi he dedicat:
    - Per una banda, s'ha creat una aplicació molt intuïtiva i simple d'utilitzar, amb uns colors que segueixen el *branding* de l'empresa i que fan que encaixi molt bé amb tot el producte que ofereixen. No confondre **simple** amb **senzilla**, tota la interfície s'ha dissenyat per tal que l'usuari vegi el màxim d'informació possible dins d'una mateixa vista (com podria ser la

de *Dashboard*) sense que això faci desorientar-lo ni crear una sensació de no saber on trobar les coses ni navegar-hi.

- També s'ha integrat la *API* de *Google Maps* a tota la gestió dels *Events* de l'aplicació. El client ho va sol·licitar, ja que *Maps* és una aplicació molt utilitzada avui en dia, i faria que els treballadors s'adaptessin més ràpidament a aquesta nova secció que crearíem.
- Per acabar, he aconseguit implementar el *real-time* a dins l'aplicació, tal com ho demanava (no obligatòriament) el client. Aquesta podria dir que ha estat la tasca més extensa i complicada de realitzar, però personalment, tot i no ser obligatòria, és la que li dóna més dinamisme i sensació de tenir davant una aplicació activa i competent.

Donats per assolits tots els anteriors requeriments, a nivell personal, considero que ha estat un projecte molt complet, més del que m'esperava a l'inici de tot quan vaig plantejar la idea. Havent de dissenyar els tres camps (DB, API, WEB) jo mateix, m'ha fet entendre i aprendre molts conceptes nous i, sobretot, a treballar amb 3 àmbits diferents i aprendre a formalitzar la connexió entre ells.

Com vaig dir a l'inici d'aquesta memòria, el fet de voler crear aquesta aplicació web per Startups va venir-me a la ment per moltes coses que havia trobat a faltar a les meves anteriors feines com a programador i que, arribat aquest moment, volia intentar solucionar i plasmar en una nova eina que soluciones tots aquests detalls. Durant el transcurs de l'aplicació, he intentat rectificar aquests "errors" que havia vist anteriorment i desenvolupar parts concretes que solucionaven i donaven a l'usuari tot el que jo havia trobat a faltar en algun moment, cosa que m'ha fet aprendre totes les dificultats d'aquests processos i arribar a entendre perquè altres aplicacions utilitzades no havien pogut incorporar algunes d'aquestes funcionalitats.

Pel que fa a nivell de tecnologies utilitzades, he après moltes coses relacionades amb el desenvolupament web, les *best practices* utilitzades en aquest sector i moltes curiositats que avui en dia s'utilitzen a l'hora de desenvolupar aplicacions com aquesta. Amb la idea de voler aprendre i aprofundir en el coneixement de JavaScript, he optat per llegir un llibre molt famós anomenat *Eloquent JavaScript* [16], el qual m'ha ajudat a solucionar

molts problemes de lògica i implementació i, fins i tot (cosa que fa que encara estigui més satisfet d'aquest projecte), m'ha servit per accelerar les meves assignacions diàries a la feina, ja que treballo programant en aquest llenguatge. Ha estat un èxit en ambdues parts.

De cara al treball futur, personalment ho explicaria desenvolupant dues idees que tinc en ment i han anat agafant pes al llarg de tot el procés:

- A la creació i edició dels *Events*, es podria desenvolupar la selecció del lloc on es celebra l'esdeveniment amb una llibreria diferent de la de *Google Maps*, la qual no fos de pagament i permetés buscar llocs a l'instant, donant suggeriments segons les paraules entrades al buscar el carrer o restaurant i així fer una selecció totalment personalitzada. Ara mateix només disposa d'un desplegable amb els llocs que l'empresa proporciona un cop es genera el compte a l'aplicació, però en un futur, la gestió del lloc personalitzada tindria molt més sentit, sobretot de cara a l'experiència d'usuari.
- Per altra banda, crear una aplicació mòbil que adaptés totes les funcionalitats a una pantalla més petita seria un gran avanç en molts sentits. Avui en dia, el mòbil és el dispositiu més utilitzat a nivell d'aplicacions i interacció entres les persones i és totalment lògic fer una adaptació de l'aplicació per aquestes plataformes. Ja sigui per la secció de missatgeria com per la de publicacions, però tenir una versió dissenyada específicament per treballar en dispositius mòbils seria una peça fonamental en tot l'ecosistema que he dissenyat. A part, la part "difícil" de la lògica i gestió de dades ja està creada, ja que la *API* està totalment desacoblada de l'aplicació web actual, fent que tinguem llibertat absoluta per utilitzar-la en aquest nou projecte.

8. Glossari

- **API RESTFul**: és una interfície de programació la qual defineix un seguit de funcions i procediments a través dels quals dos programes o llocs web intercanvien dades, funcionant sobre el protocol estàndard HTTP.
- **Back-End**: especialització dins del sector de la informàtica que es dedica a treballar desenvolupant codi que s'executa a la part del servidor i, a vegades, amb temes relacionats amb la base de dades.
- **Front-End**: molt semblant al Back-End, però en aquest cas seria més enfocat a la part del codi que s'executa a la part del client (aplicació web, mòbil..).
- **Full-Stack**: és la persona que treballa desenvolupant el codi que s'executa tant al servidor com al client.
- **JSON**: és un format basat en text estàndard per a representar dades estructurades en la sintaxi d'objectes de JavaScript.
- **JWT (JSON Web Token)**: és un estàndard basat en JSON per tal de generar tokens que serveixin per la l'autenticació entre aplicacions.
- **SCRUM**: és un procés en el qual s'apliquen de manera regular un conjunt de bones pràctiques per treballar en equip i obtenir el millor resultat possible d'un projecte.
- **SPA (Single Page Application)**: és una aplicació web que carrega tot el contingut dinàmicament sobre ella mateixa, és a dir, que només es demana el primer cop que es fa la connexió inicial al servidor. A partir de llavors, JavaScript és l'encarregat de canviar tot el que es mostra en cada moment.
- **Sprint**: cada una de les iteracions que formen el cicle de desenvolupament d'un projecte sota el procés d'SCRUM.
- **Startup**: és un terme utilitzat per a definir a aquelles empreses que es troben en edat primerenca o nova creació i presenten grans possibilitats de creixement.
- **UX (User eXperiencie)**: la percepció deixada en la ment d'un usuari després d'una sèrie d'interaccions entre usuaris, dispositius i esdeveniments - o una combinació d'aquests.

9. Referències

9.1. Tecnologies, software i llibreries

- [1] Visual Studio Code

Enllaç (visitat el dia 18 de Febrer de 2019): https://code.visualstudio.com/

Visual Studio Code és un editor de codi font multi plataforma i gratuït desenvolupat per Microsoft.

- [2] Studio 3T

Enllaç (visitat el dia 18 de Febrer de 2019): https://studio3t.com/

Interfície gràfica d'usuari que ens permet treballar amb MongoDB. D'aquesta manera és molt més senzill tant visualitzar com tractar tot el conjunt de dades que he generat.

- [3] MongoDB (v.4.0.3)

Enllaç (visitat el dia 19 de Febrer de 2019): https://www.mongodb.com/

Sistema de base de dades NoSQL orientat a documents, escalable, sense esquemes i alt rendiment. És el que he escollit per desenvolupar i guardar totes les dades de l'aplicació.

- [4] Node.js (v11.6.0)

Enllaç (visitat el dia 19 de Febrer de 2019): https://nodejs.org/en/

Node.js és l'entorn de programació que he escollit per a desenvolupar la part del servidor del projecte (*API*). És utilitzat majoritàriament a la part de Back-End, programat amb JavaScript i altament escalable, basant-se en una arquitectura orientada a esdeveniments i asíncrona.

- [5] React (v16.5.2)

Enllaç (visitat el dia 21 de Febrer de 2019): https://reactjs.org/

Llibreria de JavaScript creada per a dissenyar interfícies d'usuari. Gràcies a la manipulació del DOM (Document Object Model) que fa, permet crear interfícies molt optimitzades i ràpides, actualitzant en tot moment només les dades necessàries i així evitar la càrrega innecessària al navegador del client.

- [6] express (v4.16.4)

Enllaç (visitat el dia 21 de Febrer de 2019): https://www.npmjs.com/package/express

Express és un framework de Node.js dissenyar per a facilitar la creació d'aplicacions web i *APIs*. És ràpid i minimalista, fent molt senzilla la tasca de desenvolupar aplicacions altament escalables i fàcils de mantenir.

- [7] express-validator (v5.3.1)

Enllaç (visitat el dia 24 de Febrer de 2019): https://www.npmjs.com/package/express-validator

Middleware per tal de verificar totes les dades que rebem des de l'aplicació web dins la petició que fem a la *API*.

- [8] body-parser (v1.18.3)

Enllaç (visitat el dia 26 de Febrer de 2019): https://www.npmjs.com/package/body-parser

Middleware per parserjar totes les dades que conté el cos de les peticions que rebem des de la *API* i així deixar-les accessibles fàcilment a la variable *body* de tota funció que pugui accedir o tractar la petició.

- [9] bcryptjs (v2.4.3)

Enllaç (visitat el dia 2 de Març de 2019): https://www.npmjs.com/package/bcryptjs

Llibreria que permet encriptar/desencriptar les contrasenyes per tal de guardar-les a la base de dades amb un format que, en cas d'accedir-hi sense autenticació, no es poguessin desxifrar a simple vista.

- [10] Mongoose (v5.4.12)

Enllaç (visitat el dia 20 de Febrer de 2019): https://mongoosejs.com/

Llibreria ODM (Object Data Modeling) per a Node.js que permet treballar amb MongoDB. Facilita tot el tema de la comunicació amb la Base de Dades, relacions entre els diferents documents, creació de tot tipus de consultes i molt més.

- [11] create-react-app (v2.0.3)

Enllaç (visitat el dia 18 de Febrer de 2019): https://github.com/facebook/create-react-app

Llibreria que permet la creació d'aplicacions web amb React d'una manera simplificada i senzilla, la qual prepara un entorn de treball amb una configuració per defecte i optimitzada perquè ens puguem centrar a desenvolupar el que realment ens interessa.

- [12] date-fns (v1.30.1)

Enllaç (visitat el dia 3 d'Abril de 2019): https://www.npmjs.com/package/date-fns

Llibreria que permet treballar amb les dates que tenim guardades a la base de dades tant al navegador com a NodeJS. Té un conjunt de mètodes molt fàcils d'implementar i a l'hora molt útils, com a comparació entre dues dates, poder formatejar les dates a un nivell molt precís o fins i tot fer tot tipus de comparacions amb el moment actual respecte una data donada (és avui? és passat?)

- [13] socket.io i socket.io-client (v2.2.0)

Enllaç Socket.IO: https://www.npmjs.com/package/socket.io

Enllaç Socket.IO-Client: https://www.npmjs.com/package/socket.io-client

(Enllaços visitats el dia 15 d'Abril de 2019)

Aquest és un cas una mica especial ja que tenim dues llibreries a instal·lar. La primera és exclusiva per la part de servidor, en el nostre cas per la *API* creada amb *NodeJS*. S'encarrega de gestionar les connexions dels usuaris, guardant els *sockets* i els identificadors que serviran per saber on enviar les dades necessàries.

L'altre cas és el del client, que s'encarrega de connectar-se al servidor i establir-hi una connexió per tal de rebre les dades que es vagin generant. Tenen una estructura una mica diferent a nivell de les funcions, però la base és la mateixa al final.

Gràcies a aquestes dues llibreries, com deia, és possible enviar dades a temps real del servidor al client sense haver d'estar peticions preguntant si el recurs que necessitem ja està disponible.

9.2. Enllaços visitats:

- [14] Salaris dels enginyers a Espanya

  Pàgines web visitades el dia 3 de Maig de 2019.

  - https://www.indeed.es/salaries/Front-end-developer/a-Salaries
  - https://www.infojobs.net/ofertas-trabajo/full-stack
  - https://www.skylabcoders.com/es/-cu%C3%A1l-es-el-salario-de-un-programador-_13037
  - https://ayudawp.com/cuanto-cobra-un-desarrollador-web-en-espana-y-en-el-resto-de-europa/
  - https://www.domestika.org/es/forums/5-programacion-cliente/topics/58214-cuanto-cobra-un-programador-de-front-end
  - https://computerhoy.com/noticias/software/sueldo-programadores-descubierto-31147

- [15] Càlcul dels costos mensuals dels servidors:

  Enllaços consultats el dia 14 de Maig de 2019.

  - AWS: https://calculator.s3.amazonaws.com/index.html
  - Google Cloud: https://cloud.google.com/products/calculator/?

- Anàlisi de les diferents tecnologies utilitzades:

  Enllaços consultats el dia 16 de Maig del 2019.

  - NoSQL vs SQL: https://www.youtube.com/watch?v=ZS_kXvOeQ5Y

- Frameworks Front-End: https://www.youtube.com/watch?v=SWZ_4YBFBhs
- ExpressJS vs Laravel: https://www.slant.co/versus/1277/9622/~express-js_vs_laravel-5
- Angular al 2019: https://www.youtube.com/watch?v=Dk6sphzmKO8
- React al 2019: https://www.youtube.com/watch?v=XWfHTLm6ECQ
- Vue al 2019: https://www.youtube.com/watch?v=SC74AXdOCh4

9.3. Bibliografia

- [16] Eloquent JavaScript

Lectura online: https://eloquentjavascript.net/Eloquent_JavaScript.pdf

Llibre que explica en profunditat el llenguatge JavaScript, des de lo més bàsic a un nivell molt avançat.